2019 年度 修士論文

メニーコア CPU 上での
粗粒度タスクへの計算機資源動的割当手法
―完全準同型暗号を用いた
クライアントーサーバアプリケーションの
平均レイテンシ短縮を目指して―

DAMCREM: Dynamic Allocation Method of Computation REsource to Macro-Tasks on Many-Core CPU - Aiming to Decrease Average Latency of Client-Server Application with Fully Homomorphic Encryption -

提出日： 2020 年 01 月 29 日
指導： 山名 早人 教授

早稲田大学 大学院基幹理工学研究科 情報理工・情報通信専攻
学籍番号：5118F055-5

鈴木　拓也

概　要

近年，Web アプリケーションに代表されるクライアント—サーバアプリケーションが多数利用されている．サービス品質保証の一つとして，ジョブのレイテンシに対して，ソフトリアルタイムアプリケーションとしてのデッドラインが設定される．しかし，クライアントにとっては，デッドラインを守るだけではなく，ジョブのレイテンシができるだけ短いことが望ましい．例えば，暗号化したまま加算や乗算が実行可能となる暗号方式である完全準同型暗号を用いたアプリケーションは，平文でのアプリケーションと比べて実行時間が長い．そのため，クライアントができるだけ早く実行結果を得られるようにする必要がある．そこで，本論文では，メニーコア CPU 上での完全準同型暗号アプリケーションを対象に，ジョブを構成する個々のマクロタスクに含まれる処理に対して，動的にスレッド数を選択して実行することでジョブのレイテンシを短縮する手法 DAMCREM を提案する．具体的には，(1) 個々のマクロタスクに対して静的解析を実施して，使用スレッド数の候補を取得しておき，(2) 実行時に，実行中マクロタスクや実行待ちマクロタスクの処理内容や数に応じて，次に実行するマクロタスクを，候補から選択したスレッド数を用いて並列実行する．評価実験では，Xeon Phi 7230F の 1 スレッドでジョブやマクロタスクの実行を管理し，合計 63 スレッドを用いてマクロタスクの実行した場合，各マクロタスクに対して使用するスレッド数を静的に決定するナイーブ手法と比較し，負荷が高い状況では DAMCREM が最も平均レイテンシが低いという結果が得られた．また，負荷が低い状況では，DAMCREM は，マクロタスクに常に 1 スレッドのみ割り当てるナイーブ手法よりも平均レイテンシを短縮することができたが，常に 1 つのマクロタスクに多数のスレッドを割り当てるナイーブ手法より平均レイテンシが高かった．

目次

第1章　はじめに

現在，Web アプリケーションといった多くのクライアントーサーバアプリケーションが利用されている．クライアントーサーバアプリケーションでは，サービス品質保証として，様々な基準を設けている[1]．その基準の一つとして，サーバがクエリを受信してからアプリケーションの実行結果を送り返すまでの時間(以下，ジョブのレイテンシ)にソフトリアルタイムアプリケーションにおけるデッドラインを設けている．ジョブのレイテンシがデッドラインを超える場合，クライアントの利便性や満足度が低下することになる．一方で，ジョブのデッドラインが長い場合では，デッドラインを守るだけではなく，できるだけジョブのレイテンシが短いことが，クライアントにとって望ましい．また，ジョブごとのレイテンシを短くすることでネットワーク上での予期せぬ遅延に対応できるようになる．

ジョブのレイテンシを短縮する方法として，ジョブを構成するタスクの一部を並列に実行することが挙げられる．複数のタスクを並列に実行することで，計算資源の使用率を向上させることができる．しかし，タスク並列化を行うと，同期処理といったオーバヘッドが生じる．並列化の粒度が細かいほど，オーバヘッドは大きくなる．そして，クエリが高頻度で到着する状況において，細粒度の並列化を行うと，タスク並列化によるオーバヘッドによって単位時間あたりに実行完了できるジョブ数が減少し，スループットが低下する．スループットが低下すると，実行中ジョブ数が増加していき，ジョブのレイテンシが長くなる．このため，並列化の粒度を負荷に応じて決定する必要がある．そこで，ジョブ単位で使用する CPU コア数を決定する手法が提案されている．例えば，未使用 CPU コアを管理するモジュールや，ジョブごとにジョブの実行を管理するモジュールを用意し，ジョブに割り当てる CPU コア数をモジュール間で調整するといった手法が存在する[2], [3]．同時に実行するジョブ数が増えるほど，増加したジョブを管理するモジュールによる CPU コアの利用やモジュール間通信が原因で性能が劣化するおそれがある．

他にも，ハードリアルタイムアプリケーションでは，実行時に動的にタスク統合やタスク分割を行うといった手法が研究されている．例えば，ある期間における各 CPU コアに対して，どのように実行すべきタスクをまとめたり，期間に収まらないタスクを 2 つに分割したりしてから CPU コアに割り当てれば，デッドラインに間に合わせることができるか，という問題を解決するスケジューリング手法が存在する[4]．同様にして，平均レイテンシを短縮できるように，ジョブの平均レイテンシを短縮できるようにタスクの統合を行う手法が考えられる．しかし，頻繁に fork-join が発生する処理を対象とした場合，fork した 1 つの処理を 1 個のタスクとすると，一度のタスク統合の対象とすべきタスク数が膨大になり，タスク統合におけるオーバヘッドが大きくなる．

そこで，本論文では，様々な負荷に対応できるように，アプリケーションを粗粒度で分割したマクロタスクで構成し，マクロタスクごとに割り当てる計算資源を動的に決定することで，計算資源割り当てにおけるオーバヘッドを軽量に保ちつつ，レイテンシを短縮することを目指す．アプリケーションとして，完全準同型暗号を用いたソフトリアルタイムアプリケーションを対象とする．完全準同型暗号とは，暗号化したまま加算(以下，準同型加算)や乗算(以下，準同型乗算)が実行可能である暗号方式である．準同型加算や準同型乗算の他に，暗号文のパラメータを調整す

ることを目的とした *Relinearization* や *Rescaling*, *Bootstrapping* と呼ばれる準同型演算も存在する．整数型や浮動小数点型での計算と比べて，完全準同型暗号上での計算は，時間計算量及び空間計算量が大きいという問題点が存在する．この問題点を解決するために，メニーコア CPU[1] [2]や GPU[5–7]や FPGA[8]を用いた並列処理による高速化が行われている．

本論文では，アプリケーションをマクロタスクグラフで表す．マクロタスクを，計算資源の割当や実行順序に対するスケジューリング対象の最小単位とする．また，1 つのマクロタスクは，1 つ以上の準同型演算で構成されているとし，データ依存関係がないマクロタスク同士を並列で実行する．その上で，マクロタスクごとに割り当てるスレッド数の候補を静的に決定し，マクロタスクの実行開始時に候補から動的に選択する．つまり，状況に応じて，マクロタスクを 1 スレッドで実行するか，マクロタスクに含まれる各準同型演算を構成する処理を分割して複数のスレッドで実行する．なお，完全準同型暗号は，暗号文を条件とした条件分岐を行うことができない．また，一定の制約下で，依存関係にある一連の準同型演算の実行順序を変更することができる．完全準同型暗号を用いたアプリケーションを対象とした研究では，静的に実行順序を変更することを対象とした研究は存在する[9–13]が，動的な実行順序の変更は行われていない．したがって，本論文では，マクロタスクグラフの形状は静的に決定し，動的には変更しないとする．

本論文での具体的な問題設定を示す．

- クライアント：複数台
- サーバ：メニーコア CPU を搭載した計算機 1 台（GPU や FPGA を搭載せず，マルチ CPU や複数台の計算機を用いた分散処理を対象としない）
- クエリ：1 台のクライアントからサーバへ送る 1 まとまりの入力データ
- サーバが実行するアプリケーションの種類数：1 種類
- サーバが実行するアプリケーションのマクロタスクグラフ：クエリに依らず常に同じ．ジョブ間の依存は無い．
- クエリ到着間隔：クエリ到着頻度がポアソン分布に従う．ただし，平均到着間隔をサーバが知ることはできない．
- 計算資源の制約：消費電力を考慮せず，サーバの計算資源を可能な限り利用する．
- ジョブの大きさ：複数のジョブを 1 つのメニーコア CPU で同時に実行可能な規模

上記の条件下で，ジョブのレイテンシをできるだけ低く保つことが目的である．なお，本論文では，メニーコア CPU を対象とするが，マルチ CPU や複数台のサーバを用いた分散処理には，提案手法を拡張することで対応可能である．また, GPU や FPGA では, 搭載された大量の演算器に,

[1] HEAAN，https://github.com/snucrypto/HEAAN
[2] PALISADE，https://git.njit.edu/palisade/PALISADE

可能な限り細かく分割されたタスクを割り当てて実行するため，マクロタスクに対して計算資源を動的に割り当てる効果はメニーコア CPU と比べて小さい．したがって，本論文では GPU や FPGA を対象としない．

本論文では，クエリの到着時刻に依存して負荷が変化する状況において，ジョブの平均レイテンシを短縮するために，負荷に応じてマクロタスクを，事前に用意した候補から適切なスレッド数を動的に選択して実行する手法を提案する．ただし，計算資源の限界を超える負荷は対象外とする．具体的には，マクロタスクに対して，ある時点で実行可能なマクロタスクの実行順序とマクロタスクごとに割り当てるスレッド数を動的に決定する．マクロタスクの実行順序は，(1) ジョブ間では，クエリ到着時刻が早いジョブを優先し，(2) ジョブ内では，各マクロタスクの実行時間を用いて設定した優先度に基づいて，決定する．そして，各マクロタスクに対して，使用スレッド数の候補を事前に用意しておき，実行時に，利用可能な計算資源と実行待ちマクロタスクの処理内容や数から適切なスレッド数を候補から選択し，実行する．

本論文は以下の構成である．第 2 章で完全準同型暗号について説明する．第 3 章で完全準同型暗号を用いたアプリケーションやクライアント—サーバアプリケーションに対する高速化手法や，クライアント—サーバアプリケーションやタスク粒度に関する関連研究を示す．そして，第 4 章で問題をモデル化し，提案手法を説明する．第 5 章で実施した評価実験を示し，実験結果に対する考察を第 6 章で述べる．第 7 章で今後の課題を述べた後，第 8 章でまとめる．

第2章　　完全準同型暗号の概要

本章では，完全準同型暗号について説明する．Ring-LWE 問題に基づく完全準同型暗号[14]の方式の 1 つに CKKS 方式[15–17]と呼ばれる方式がある．CKKS 方式では，スケール値と呼ばれる，小数部分の桁数を管理する値を平文及び暗号文のパラメータの 1 つとして用いることで，固定小数点で表現された複素数を暗号化し，暗号化したまま計算することができる．CKKS 方式では，他の方式とは異なり，複素数に対する高次の多項式近似計算が可能となるため，機械学習や数値計算などのアプリケーションに利用される．また，Ring-LWE 問題に基づく完全準同型暗号の方式には，多倍長整数を用いる方法と剰余系表現を用いる方法[18]の 2 種類の方法が存在する．剰余系表現を用いる方法は並列処理に適している．したがって，本論文では，剰余系表現を利用した CKKS 方式[17]を用いる．また，準同型暗号ライブラリとして SEAL[19]のバージョン 3.3.2 を用いる．なお，剰余系表現を用いる場合，平文及び暗号文に対して数論変換を適用することで，準同型乗算の実行速度を向上することができる．SEAL ライブラリでは，平文及び暗号文は，数論変換後の値を保持している．2.2 節で説明する 1 つの暗号文を入力とする *Relinearization* 及び *Rescaling* においては，数論変換後の値では理論上正しく計算できないため，適宜逆変換と変換をした上で処理を行う．*Relinearization* 及び *Rescaling* の出力暗号文は，数論変換後の値を保持している．本論文においても，SEAL ライブラリと同様に扱う．

2.1 完全準同型暗号のデータ構造

本節では，平文と暗号文のデータ構造について説明する．まず，複素数配列$\boldsymbol{x}$に対してエンコードと呼ばれる処理を行うことで平文 $\tilde{\boldsymbol{x}}$を生成する．逆に，平文 $\tilde{\boldsymbol{x}}$に対して，デコードと呼ばれる処理を行うことで，複素数配列$\mathrm{Dcd}(\tilde{x})$を生成する．平文を $\tilde{\boldsymbol{x}}$を暗号化することで暗号文$\mathrm{Enc}(\tilde{x})$が得られ，暗号文$\mathrm{Enc}(\tilde{x})$を復号することで，平文$\mathrm{Dec}\big(\mathrm{Enc}(\tilde{x})\big)$が得られる．ここで，エンコードの過程で，初期スケール値 $\bar{q}$を乗算し，整数に丸めることで，複素数配列の小数部分を平文上で扱えるようにする．丸め誤差が生じるため，元の複素数配列$\boldsymbol{x}$と平文 $\tilde{\boldsymbol{x}}$をデコードして得られた複素数配列$\mathrm{Dcd}(\tilde{x})$の値は一致しないことがある．また，平文と暗号文は，次数Nの多項式環を用いて表される．次数Nはアプリケーションごとに静的に決定される値であり，$N \in \{2^{12}, 2^{13}, \cdots, 2^{16}\}$が主に使用される．次数$N$に応じて，1 つの平文に$\frac{N}{2}$個の複素数をエンコードすることができる．さらに，剰余系表現を用いることで，中国剰余定理により，多項式環の各係数を1[word]に収まる値を複数用いることで表現できる．そして，平文は$l+1$個の多項式環を 1 組用いて構成され，暗号文は$l+1$個の多項式環を 2 組以上用いて構成される．すなわち，平文は$(l+1)\times N$の 2 次元配列，暗号文は$S\times(l+1)\times N$の 3 次元配列としてそれぞれ表すことができる．ただし，lは 0 以上の整数であり，lを「レベル」と呼び，平文 $\tilde{\boldsymbol{x}}$のレベルを$\mathrm{Level}(\tilde{x})$，暗号文$\mathrm{Enc}(\tilde{x})$のレベルを$\mathrm{Level}\big(\mathrm{Enc}(\tilde{x})\big)$と表記する．レベルは，対象の平文 $\tilde{\boldsymbol{x}}$もしくは暗号文$\mathrm{E}(\tilde{x})$に対して適用できる *Rescaling* と *Simple Modulus Reduction*[15]の残り回数である．*Rescaling* と *Simple Modulus Reduction* の詳細は，2.2 節で示す．*Rescaling* か *Simple Modulus Reduction* を適用するたびに，対象の平文もしくは暗号文のレベ

ルが 1 減少する．また，Sは 2 以上の整数であり，本論文では，このSを「暗号文の長さ」と呼び，暗号文$\mathrm{Enc}(\tilde{x})$の長さを$\mathrm{Size}\big(\mathrm{Enc}(\tilde{x})\big)$と表記する．平文は常に長さが 1 である．以降，「2 つの平文もしくは 2 つの暗号文のパラメータが同じである」ということは，「平文もしくは暗号文を構成するスケール値とレベル，長さが同じである」とし，「2 つの平文もしくは 2 つの暗号文が同じである」ということは，「平文もしくは暗号文を構成する多項式環の各係数の値とパラメータが同じである」とする．図 2.1 と図 2.2 に平文と暗号文のそれぞれのデータ構造を示す．$\hat{x}_{i,j,k}, (0 \le i < S, 0 \le j \le l, 0 \le k < N)$は多項式環の係数である．1 つの暗号文のデータサイズは数十[KB]から数十[MB]である．暗号文の次数Nやレベルlといった暗号文のパラメータが大きいほど空間計算量が大きく，CPU のキャッシュに乗り切らない場合もある．

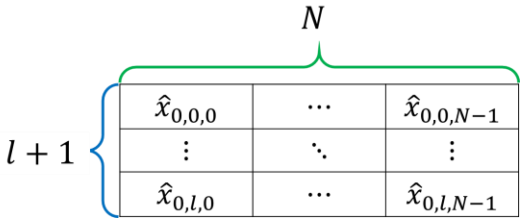

図 2.1 平文のデータ構造

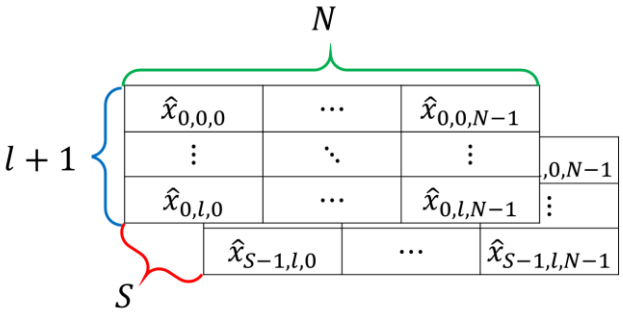

図 2.2 暗号文のデータ構造

多くの完全準同型暗号を用いたアプリケーションでは，暗号化された値を条件とした条件分岐を行うことができない[3]ため，実行する準同型演算と順番を予め決定することができる．つまり，各平文や各暗号文に適用される *Rescaling* 及び *Simple Modulus Reduction* の回数を予め知ることができる．したがって，各平文や各暗号文に適用される *Rescaling* 及び *Simple Modulus Reduction* の回数の内，最大の値をレベルの初期レベルLとすることで，アプリケーションを実行することができる．ただし，安全性の観点から，多項式環の次数Nと初期スケール値 $\bar{q}$に応じて，初期レベルLの上限が存在する．初期レベルLよりも多くの回数の *Rescaling* 及び *Simple Modulus Reduction* を適用するためには，*Rescaling* によって減少したレベルの値を増やす準同型演算である *Bootstrapping* が必要である．この *Bootstrapping* は他の準同型演算と比べて特に実行時間が長いため，できるだけ使用しないことが望ましい．本論文での評価実験では，対象とするアプリケーションにおいて各平文や各暗号文に対して適用される *Rescaling* 及び *Simple Modulus Reduction* の回数の内，最大回数は，多項式環の次数Nに対して十分安全な初期レベルL以下となるように設計したため，

[3] もし条件分岐が可能であるとすると，暗号化された値を知ることができてしまうため，条件分岐はできない．

Bootstrapping は使用しない．

2.2 本論文で使用する準同型演算

本論文では，表 2.1 に示す準同型演算を用いる．図 2.3～図 2.11 に各準同型演算の演算過程を図 2.1 及び図 2.2 を用いて示す．ただし，暗号化する際に，ノイズと呼ばれる乱数値が付与される．したがって，同じ平文 $\tilde{x}$を暗号化する度に，多項式環の各係数の値が異なる暗号文が得られる．そして，暗号文$\mathrm{Enc}(\tilde{x})$を復号して得られる平文 $\mathrm{Dec}\big(\mathrm{Enc}(\tilde{x})\big)$は $\tilde{x}$とは異なる．なお，2 つの暗号文を入力とする準同型乗算では，入力暗号文が同じであれば，準同型乗算を構成する一部の処理を省略することができる．

長さについて説明する．2 つの暗号文を入力とする準同型乗算を適用することで，長さが増加し，*Relinearization* を適用すると長さが 1 減少する．暗号文の長さが大きいほど，各準同型演算における計算量が増加する．

スケール値とレベルについて説明する．準同型乗算を適用することで，出力暗号文のスケール値が，入力の平文もしくは暗号文のスケール値の積となる．スケール値が初期スケール値 $\bar{q}$と異なると，デコード時に正しい値が得られないことがある．そのため，*Rescaling* を適用して，レベルを 1 減少させて，スケール値を$\bar{q}$で除算することで，適切なスケール値にする．例えば，入力の平文もしくは暗号文のスケール値を共に $\bar{q}$とすると，準同型乗算の出力暗号文のスケール値は，$\bar{q}^2$である．ここで，*Rescaling* を適用すると，スケール値が $\frac{\bar{q}^2}{\bar{q}} = \bar{q}$となる．したがって，復号後に得られた平文に対してデコードした際に，正しい値を得ることができる．なお，2 つの暗号文を入力とする準同型乗算適用後は，*Relinearization* の後に *Rescaling* を適用する．一方，*Simple Modulus Reduction* は，平文及び暗号文の持つスケール値を変化せずにレベルを 1 減少させる準同型演算であり，準同型加算や準同型乗算への入力となる平文や暗号文のレベルを合わせるために使用される．*Simple Modulus Reduction* は，実装によっては準同型加算や準同型乗算内部で適宜適用する場合がある．しかし，SEAL ライブラリでは，*Simple Modulus Reduction* を明示的に適用する必要があるため，本論文も明示的に適用する．

表 2.1　本論文で使用する準同型演算（その 1）

準同型演算名	入力	表記	制約
準同型加算	1 つの平文と 1 つの暗号文	$\tilde{x}\oplus\mathrm{Enc}(\tilde{y})$ $\rightarrow \mathrm{Enc}(\tilde{z})$	●$\mathrm{Dcd}(\tilde{x})+\mathrm{Dcd}\left(\mathrm{Dec}\left(\mathrm{Enc}(\tilde{y})\right)\right)\approx\mathrm{Dcd}\left(\mathrm{Dec}\left(\mathrm{Enc}(\tilde{z})\right)\right)$ ●$\mathrm{Level}\left(\mathrm{Enc}(\tilde{z})\right)=\mathrm{Level}(\tilde{x})=\mathrm{Level}\left(\mathrm{Enc}(\tilde{y})\right)$ ●$\mathrm{Size}\left(\mathrm{Enc}(\tilde{z})\right)=\mathrm{Size}\left(\mathrm{Enc}(\tilde{y})\right)$
	2 つの暗号文	$\mathrm{Enc}(\tilde{x})\oplus\mathrm{Enc}(\tilde{y})$ $\rightarrow \mathrm{Enc}(\tilde{z})$	●$\mathrm{Dcd}\left(\mathrm{Dec}\left(\mathrm{Enc}(\tilde{x})\right)\right)+\mathrm{Dcd}\left(\mathrm{Dec}\left(\mathrm{Enc}(\tilde{y})\right)\right)\approx\mathrm{Dcd}\left(\mathrm{Dec}\left(\mathrm{Enc}(\tilde{z})\right)\right)$ ●$\mathrm{Level}\left(\mathrm{Enc}(\tilde{z})\right)=\mathrm{Level}\left(\mathrm{Enc}(\tilde{x})\right)=\mathrm{Level}\left(\mathrm{Enc}(\tilde{y})\right)$ ●$\mathrm{Size}\left(\mathrm{Enc}(\tilde{z})\right)=\mathrm{Size}\left(\mathrm{Enc}(\tilde{x})\right)=\mathrm{Size}\left(\mathrm{Enc}(\tilde{y})\right)$
準同型乗算	1 つの平文と 1 つの暗号文	$\tilde{x}\otimes\mathrm{Enc}(\tilde{y})$ $\rightarrow \mathrm{Enc}(\tilde{z})$	●$\mathrm{Dcd}(\tilde{x})\times\mathrm{Dcd}\left(\mathrm{Dec}\left(\mathrm{Enc}(\tilde{y})\right)\right)\approx\mathrm{Dcd}\left(\mathrm{Dec}\left(\mathrm{Enc}(\tilde{z})\right)\right)$ ●$\mathrm{Level}\left(\mathrm{Enc}(\tilde{z})\right)=\mathrm{Level}(\tilde{x})=\mathrm{Level}\left(\mathrm{Enc}(\tilde{y})\right)$ ●$\mathrm{Size}\left(\mathrm{Enc}(\tilde{z})\right)=\mathrm{Size}\left(\mathrm{Enc}(\tilde{y})\right)$
	2 つの暗号文	$\mathrm{Enc}(\tilde{x})\otimes\mathrm{Enc}(\tilde{y})$ $\rightarrow \mathrm{Enc}(\tilde{z})$	●$\mathrm{Dcd}\left(\mathrm{Dec}\left(\mathrm{Enc}(\tilde{x})\right)\right)\times\mathrm{Dcd}\left(\mathrm{Dec}\left(\mathrm{Enc}(\tilde{y})\right)\right)\approx\mathrm{Dcd}\left(\mathrm{Dec}\left(\mathrm{Enc}(\tilde{z})\right)\right)$ ●$\mathrm{Level}\left(\mathrm{Enc}(\tilde{z})\right)=\mathrm{Level}\left(\mathrm{Enc}(\tilde{x})\right)=\mathrm{Level}\left(\mathrm{Enc}(\tilde{y})\right)$ ●$\mathrm{Size}\left(\mathrm{Enc}(\tilde{z})\right)=\mathrm{Size}\left(\mathrm{Enc}(\tilde{x})\right)+\mathrm{Size}\left(\mathrm{Enc}(\tilde{y})\right)-1$
	2 つの暗号文 (2 乗)	$\mathrm{Enc}(\tilde{x})\otimes\mathrm{Enc}(\tilde{x})$ $\rightarrow \mathrm{Enc}(\tilde{z})$	●$\mathrm{Dcd}\left(\mathrm{Dec}\left(\mathrm{Enc}(\tilde{x})\right)\right)\times\mathrm{Dcd}\left(\mathrm{Dec}\left(\mathrm{Enc}(\tilde{x})\right)\right)\approx\mathrm{Dcd}\left(\mathrm{Dec}\left(\mathrm{Enc}(\tilde{z})\right)\right)$ ●$\mathrm{Level}\left(\mathrm{Enc}(\tilde{z})\right)=\mathrm{Level}\left(\mathrm{Enc}(\tilde{x})\right)$ ●$\mathrm{Size}\left(\mathrm{Enc}(\tilde{z})\right)=2\times\mathrm{Size}\left(\mathrm{Enc}(\tilde{x})\right)-1$

表 2.1　本論文で使用する準同型演算（その 2）

Relinearization	1 つの暗号文	$\mathrm{Relin}(\mathrm{Enc}(\tilde{x})) \rightarrow \mathrm{Enc}(\tilde{z})$	●$\mathrm{Dcd}\left(\mathrm{Dec}(\mathrm{Enc}(\tilde{x}))\right) \approx \mathrm{Dcd}\left(\mathrm{Dec}(\mathrm{Enc}(\tilde{z}))\right)$ ●$\mathrm{Level}(\mathrm{Enc}(\tilde{z})) = \mathrm{Level}(\mathrm{Enc}(\tilde{x}))$ ●$\mathrm{Size}(\mathrm{Enc}(\tilde{z})) = \mathrm{Size}(\mathrm{Enc}(\tilde{x})) - 1$ ●2 つの暗号文を入力とする準同型乗算の後に適用する.
Rescaling	1 つの暗号文	$\mathrm{Rescl}(\mathrm{Enc}(\tilde{x})) \rightarrow \mathrm{Enc}(\tilde{z})$	●$\mathrm{Dcd}\left(\mathrm{Dec}(\mathrm{Enc}(\tilde{x}))\right) \approx \mathrm{Dcd}\left(\mathrm{Dec}(\mathrm{Enc}(\tilde{z}))\right)$ ●$\mathrm{Level}(\mathrm{Enc}(\tilde{z})) = \mathrm{Level}(\mathrm{Enc}(\tilde{x})) - 1$ ●$\mathrm{Size}(\mathrm{Enc}(\tilde{z})) = \mathrm{Size}(\mathrm{Enc}(\tilde{x}))$ ●準同型乗算後に適用する(2 つの暗号文を入力とする準同型乗算の場合は，*Relinearization* の後)
Simple Modulus Reduction	1 つの平文	$\mathrm{SMR}(\tilde{x}) \rightarrow \tilde{z}$	●$\mathrm{Dcd}(\tilde{x}) \approx \mathrm{Dcd}\left(\mathrm{Dec}(\mathrm{Enc}(\tilde{z}))\right)$ ●$\mathrm{Level}(\tilde{z}) = \mathrm{Level}(\tilde{x}) - 1$ ●後続の準同型演算の入力となる平文と暗号文のレベルが異なる時に適用する
	1 つの暗号文	$\mathrm{SMR}(\mathrm{Enc}(\tilde{x})) \rightarrow \mathrm{Enc}(\tilde{z})$	●$\mathrm{Dcd}\left(\mathrm{Dec}(\mathrm{Enc}(\tilde{x}))\right) \approx \mathrm{Dcd}\left(\mathrm{Dec}(\mathrm{Enc}(\tilde{z}))\right)$ ●$\mathrm{Level}(\mathrm{Enc}(\tilde{z})) = \mathrm{Level}(\mathrm{Enc}(\tilde{x})) - 1$ ●$\mathrm{Size}(\mathrm{Enc}(\tilde{z})) = \mathrm{Size}(\mathrm{Enc}(\tilde{x}))$ ●後続の準同型演算の入力となる平文と暗号文のレベルが異なる時に適用する

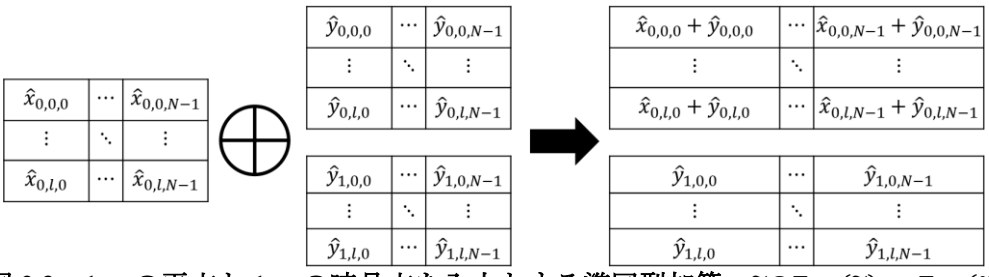

図 2.3　1 つの平文と 1 つの暗号文を入力とする準同型加算　$\tilde{x} \oplus \mathrm{Enc}(\tilde{y}) \rightarrow \mathrm{Enc}(\tilde{z})$

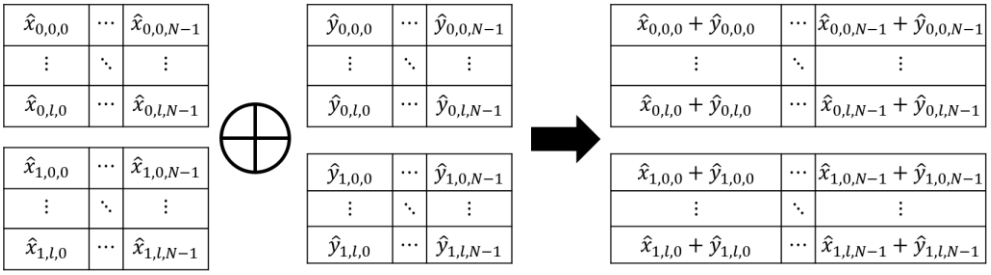

図 2.4　2 つの暗号文を入力とする準同型加算　$\mathrm{Enc}(\tilde{x}) \oplus \mathrm{Enc}(\tilde{y}) \rightarrow \mathrm{Enc}(\tilde{z})$

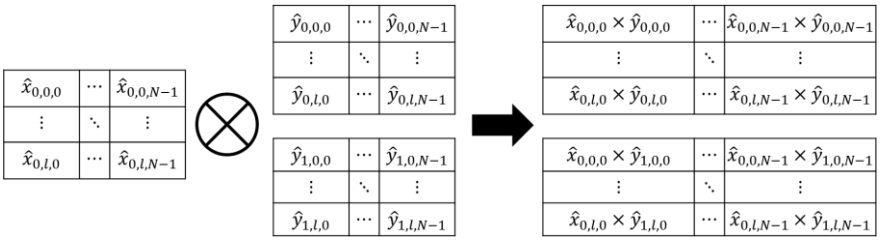

図 2.5　1 つの平文と 1 つの暗号文を入力とする準同型乗算　$\tilde{x}\otimes\mathrm{Enc}(\tilde{y}) \to \mathrm{Enc}(\tilde{z})$

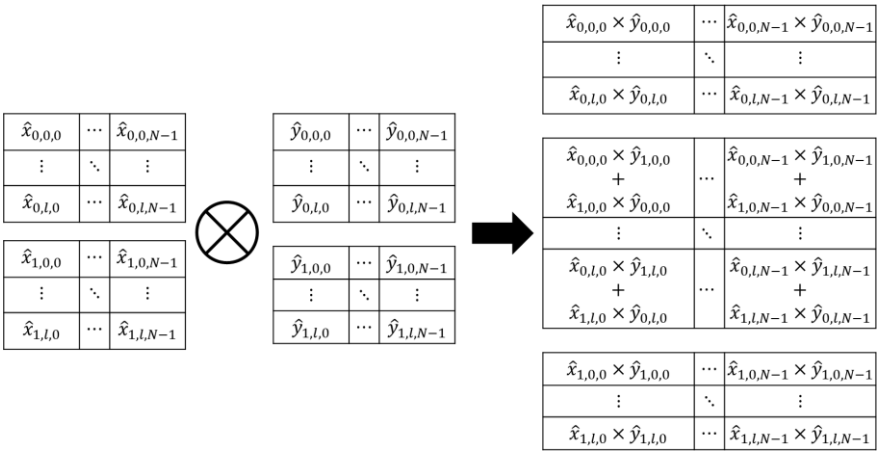

図 2.6　2 つの暗号文を入力とする準同型乗算　$\mathrm{Enc}(\tilde{x})\otimes\mathrm{Enc}(\tilde{y}) \to \mathrm{Enc}(\tilde{z})$

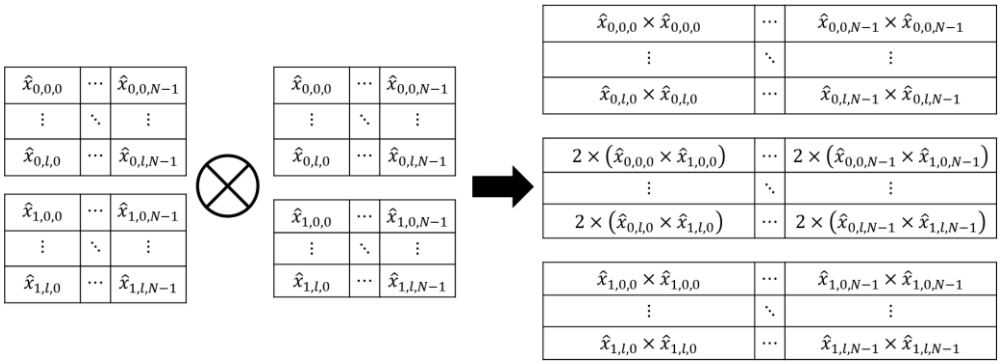

図 2.7　2 つの暗号文を入力とする準同型乗算（2 乗）　$\mathrm{Enc}(\tilde{x})\otimes\mathrm{Enc}(\tilde{x}) \to \mathrm{Enc}(\tilde{z})$

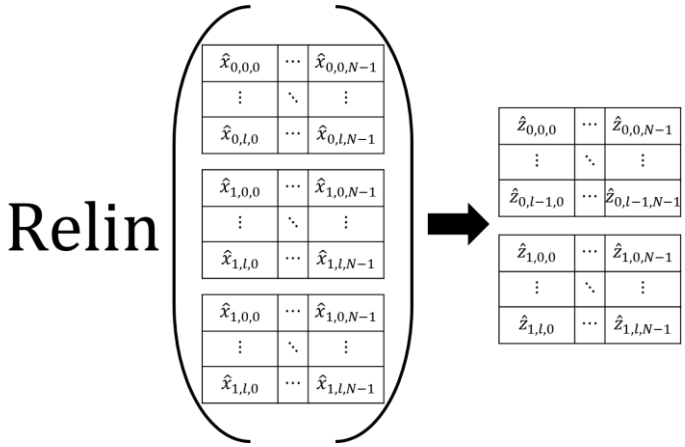

図 2.8　1 つの暗号文を入力とする *Relinearization*　$\mathrm{Relin}\big(\mathrm{Enc}(\tilde{x})\big) \to \mathrm{Enc}(\tilde{z})$

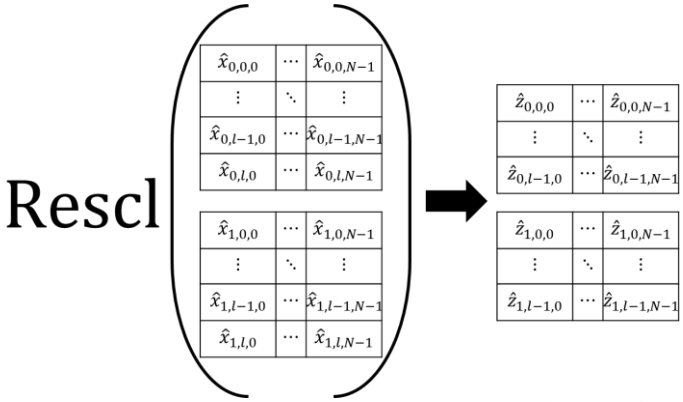

図 2.9　1 つの暗号文を入力とする *Rescaling*　$\mathbf{Rescl}(\mathbf{Enc}(\tilde{x})) \rightarrow \mathbf{Enc}(\tilde{z})$

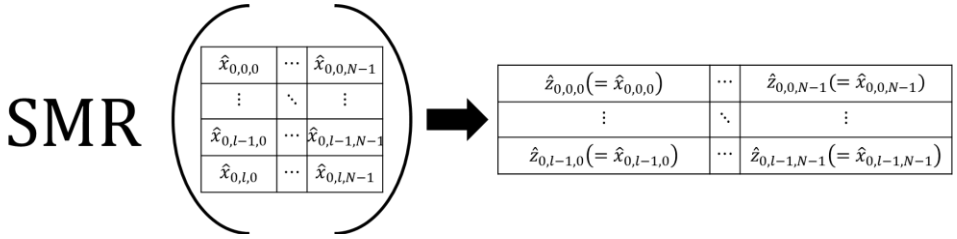

図 2.10　1 つの平文を入力とする *Simple Modulus Reduction*　$\mathbf{SMR}(\tilde{x}) \rightarrow \tilde{z}$

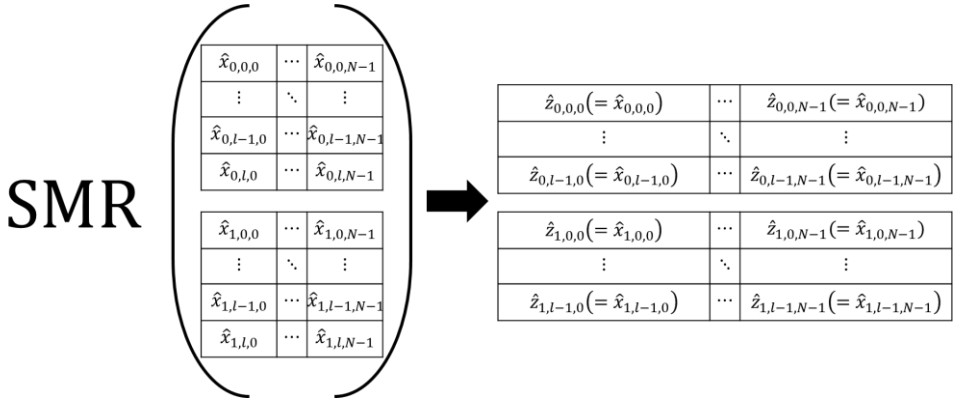

図 2.11　1 つの暗号文を入力とする *Simple Modulus Reduction*　$\mathbf{SMR}(\mathbf{Enc}(\tilde{x})) \rightarrow \mathbf{Enc}(\tilde{z})$

2.3 準同型演算の実行時間計測

本節では, 準同型演算の実行速度について示す. まず, 使用した計算機を 2.3.1 項で示す. 次に, 準同型演算をシングルスレッドで実行した時の実行時間を 2.3.2 項で示す．そして，準同型演算をマルチスレッドで実行した時の実行時間を 2.3.3 項で示す.

2.3.1　使用した計算機

本論文では，メニーコア CPU 上でのアプリケーション実行を対象としている．そこで，本論文

では，メニーコア CPU の 1 つである Xeon Phi 7230F を用いた．表 2.2 に使用した計算機の詳細を示す．Xeon Phi 7230F には，メモリモードとクラスタモードという 2 つの設定がある．メモリモードは，MCDRAM をどのように使用するかを指定するための設定であり，下記に示す 3 つの設定から 1 つを BIOS で指定する．

1. Flat : MCDRAM を DRAM とは異なる NUMA ノードとして使用する．アドレスが割り当てられるため，ユーザが DRAM と MCDRAM のどちらを使用するかを指定できる．
2. Cache：MCDRAM をラストレベルキャッシュとして使用する．MCDRAM 上に存在しないデータを DRAM から読み出す場合は，Flat モードと比べてメモリレイテンシが増加する．
3. Hybrid：Flat モードと Cache モードを組み合わせて使用する．Flat モードとする容量と Cache モードとする容量を指定できる．

クラスタモードは，CPU コアにあるタグディレクトリとメモリの分割方法を指定するための設定であり，下記に示す 5 つの設定から 1 つを BIOS で指定する．

1. All2All：タグディレクトリとメモリ間のアフィニティが存在しない．
2. Hemisphere：CPU コアとメモリを 2 分割し，領域ごとにタグディレクトリとメモリ間にアフィニティが存在する．
3. Quadrant：CPU コアとメモリを 4 分割し，領域ごとにタグディレクトリとメモリ間にアフィニティが存在する．
4. SNC[4]2：2 つの領域に分割し，それぞれを NUMA ノードとして使用する．
5. SNC4：4 つの領域に分割し，それぞれを NUMA ノードとして使用する．

本論文における実験では，メモリモードは Cache モードを使用し，クラスタモードは Quadrant モードを使用した．

[4] **S**ub **N**UMA **C**luster の略である

表 2.2　使用した計算機の情報

パーツ	項目	値
CPU	型番	Xeon Phi 7230F
	動作周波数	1.3[GHz]
	ターボブースト時の 最大動作周波数	1.5[GHz]
	物理コア数	64
	論理コア数	256
	L2 キャッシュ (2 つの物理コアで共有)	1[MB]
MCDRAM (オンチップ メモリ)	総容量	16[GB]
	チャネル数	8
	総帯域幅（理論値）	450[GB/s]
DRAM	総容量	384[GB]
	チャネル数	6
	総帯域幅（理論値）	115.2[GB/s]
ソフトウェア	OS	CentOS 7.3.1611
	GCC	6.3.1

SEAL ライブラリに対するコンパイルオプションを「-fPIC -O3 -g -DNDEBUG -maes -pthread -std=gnu++14」，SEAL のベンチマークプログラムに対するコンパイルオプションを「-maes -pthread -std=gnu++14」とした．また，第 5 章の評価実験に用いたプログラムに対するコンパイルオプションを「-O3 -DNDEBUG -Wall -fopenmp -march=native -maes -pthread -std=gnu++14」とした．

2.3.2　準同型演算のシングルスレッドでの実行時間

SEAL のベンチマークプログラムを用いて測定した各準同型演算の実行時間を表 2.3 に示す．表 2.3 に示していない，1 つの平文と 1 つの暗号文を入力とする準同型加算と *Simple Modulus Reduction* は，2 つの暗号文を入力とする準同型加算よりも実行時間が短い．なお，パラメータは，次数$N = 32{,}768$に固定し，レベル$l \in \{1, 2, \cdots, 9\}$を変数とした．

準同型加算はレベルが増加しても数[ms]で実行完了する．その次に準同型乗算の実行時間が長い．準同型乗算の内，入力が 1 つの平文と 1 つの暗号文が最も短い．入力が暗号文のみの場合は，2.2 節で示した通り，2 乗を計算する場合は一部の処理を省略できるため，入力が異なる 2 つの暗号文である準同型乗算と比べて実行時間が約 70[%]である．そして，*Relinearization* は本論文で使用する準同型演算で最も実行時間が長く，*Rescaling* はその次に実行時間が長い．一部の例外を除き，2 つの暗号文を入力とする準同型乗算を適用後に *Relinearization* を適用する必要があり，準同型乗算を適用後に *Rescaling* を適用する必要がある[5]．したがって，ジョブの実行時間を短縮する

[5] 例えば，準同型乗算適用後に暗号文のパラメータが同じ暗号文同士を入力とする準同型加算を適用する場合，*Relinearization* や *Rescaling* を，準同型乗算直後に適用する代わりに，準同型加算適用直後に適用でき，*Relinearization* や *Rescaling* の回数を削減することができる[11], [21].

ためには，*Relinearization* と *Rescaling* の実行時間を短縮する必要がある．

表 2.3　各準同型演算の実行時間

準同型演算名	レベルlでの実行時間[ms]								
	$l=1$	$l=2$	$l=3$	$l=4$	$l=5$	$l=6$	$l=7$	$l=8$	$l=9$
準同型加算 $\mathrm{E}(x)\oplus\mathrm{E}(y)\rightarrow\mathrm{E}(z)$	0.8	1.1	1.5	1.9	2.3	2.7	3.0	3.4	3.7
準同型乗算 $x\otimes\mathrm{E}(y)\rightarrow\mathrm{E}(z)$	5.6	8.6	11.4	13.8	17.6	19.6	23.0	26.2	28.3
準同型乗算 $\mathrm{E}(x)\otimes\mathrm{E}(y)\rightarrow\mathrm{E}(z)$	13.3	20.4	26.8	32.8	41.2	46.5	53.7	60.9	66.3
準同型乗算(2 乗) $\mathrm{E}(x)\otimes\mathrm{E}(x)\rightarrow\mathrm{E}(z)$	9.4	14.2	18.8	22.9	28.5	32.5	38.0	43.3	46.6
Relinearization $\mathrm{Relin}(\mathrm{E}(x))\rightarrow\mathrm{E}(z)$	69.4	113.6	167.2	226.6	307.7	381.0	463.9	565.2	668.0
Rescaling $\mathrm{Rescl}(\mathrm{E}(x))\rightarrow\mathrm{E}(z)$	31.0	52.1	74.1	95.6	121.8	138.0	155.9	175.4	194.7

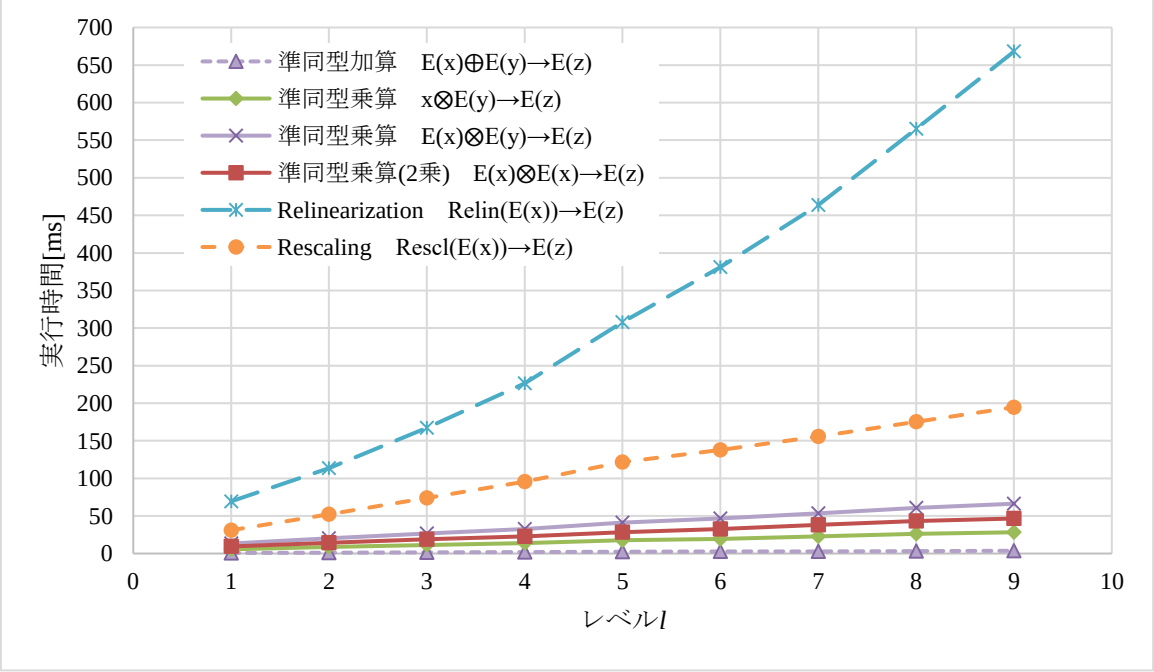

図 2.12　レベルlによる各準同型演算の実行時間の変化

2.3.3　準同型演算のマルチスレッドでの実行時間

本項では，準同型演算の内，*Relinearization* と *Rescaling* をマルチスレッドで実行した時の実行時間を示す．*Relinearization* と *Rescaling* には，それぞれ複数の for 文が存在する．具体的には，*Relinearization* は，4 つの for 文を含み，それぞれイテレーション数が$(l+1)$, $(l+2)$, 2, $2\times(l+1)$であり，*Rescaling* は，3 つの for 文を含み，それぞれイテレーション数が$2\times(l+1)$，$2\times l$，$2\times l$である．つまり，*Relinearization* と *Rescaling* はそれぞれレベルlに対して，最大$2\times(l+1)$スレッドで並列化することができる．SEAL ライブラリは *Relinearization* 及び *Rescaling* を構成する for 文の処理を並列実行しないため，pthread を用いて並列実行を行えるように改変した．そして，得られた *Relinearization* の実行時間を表 2.4 に，*Rescaling* の実行時間を表 2.5 にそれぞれ示す．ただし，*Relinearization* と *Rescaling* を構成する一部の for 文は$2\times(l+1)$スレッド未満でしか並列化できない．また，並列化できない処理も存在する．したがって，nスレッドで並列化しても実行時間は$\frac{1}{n}$倍にはならない．つまり，実行すべき準同型演算が大量に存在する場合は，1 つの準同型演算には 1 スレッドのみを使用し，複数の準同型演算を並列に実行すべきである．

レベルや準同型演算によっては，スレッド数を増やしても実行時間が短くならない場合がある．具体的には，レベル$l=5$での *Relinearization* に対して，スレッド数を 4 スレッドから 5 スレッドに増やした場合である．レベル$l=5$の場合，*Relinearization* を構成する各 for 文のイテレーション数は，6，7，2，12 である．したがって，4 スレッドで実行しても，5 スレッドで実行しても，実行時間を最小にするためには，各 for 文に対して，少なくとも 1 つのスレッドはそれぞれ 2，2，1，3 イテレーションずつ処理を行うことになる．この時，各 for 文を実行する複数のスレッドの内，最も長い実行時間がかかるスレッドで全体の実行時間が決まるため，*Relinearization* にかかる実行時間は変わらない．言い換えると，4 スレッドで実行することと比べて，5 スレッドで実行することは，1 スレッド分の計算資源を適切に使えておらず，不適切である．同様に，8〜11 スレッドも不適切なスレッド数である．また，*Relinearization* では$(l+2)$スレッドを，*Rescaling* では$(l+1)$スレッドを超えると，実行速度の向上率が低い．なお，レベル$l=1$における *Relinearization* は 2 スレッド使用時と 3 スレッド使用時に実行時間に大きな差がない．これは，スレッド生成及び同期のオーバヘッドや各スレッドのために確保するメモリが増加するためと考えられる．まとめると，使用するスレッド数はレベルlに応じて適切に選択する必要がある．

表 2.4　*Relinearization* に対するマルチスレッドでの実行時間

スレッド数	レベルlでの実行時間[ms]								
	$l=1$	$l=2$	$l=3$	$l=4$	$l=5$	$l=6$	$l=7$	$l=8$	$l=9$
1	69.4	113.6	167.2	226.6	307.7	381.0	463.9	565.2	668.0
2	40.9	67.6	101.8	129.3	179.1	217.3	279.6	320.3	398.6
3	39.1	52.2	75.2	95.9	133.2	161.2	186.1	240.3	275.1
4	30.8	44.1	61.4	86.2	94.9	115.7	165.4	187.2	225.3
5		45.1	47.6	72.1	97.0	109.5	126.7	134.6	188.2
6		35.7	48.4	53.7	84.8	107.4	119.5	130.6	144.1
7			49.8	53.6	60.6	95.4	117.1	131.7	142.0
8			42.3	53.7	57.2	65.0	106.1	128.5	136.9
9				53.6	58.9	64.2	70.6	115.7	141.4
10				47.1	60.4	65.6	70.4	74.5	125.0
11					58.4	65.2	70.3	73.9	78.9
12					52.0	64.8	69.0	74.9	80.5
13						64.7	68.5	73.5	79.4
14						56.3	68.5	74.8	82.5
15							68.9	75.4	80.0
16							62.3	73.5	79.4
17								73.7	79.6
18								67.1	79.2
19									80.8
20									74.4

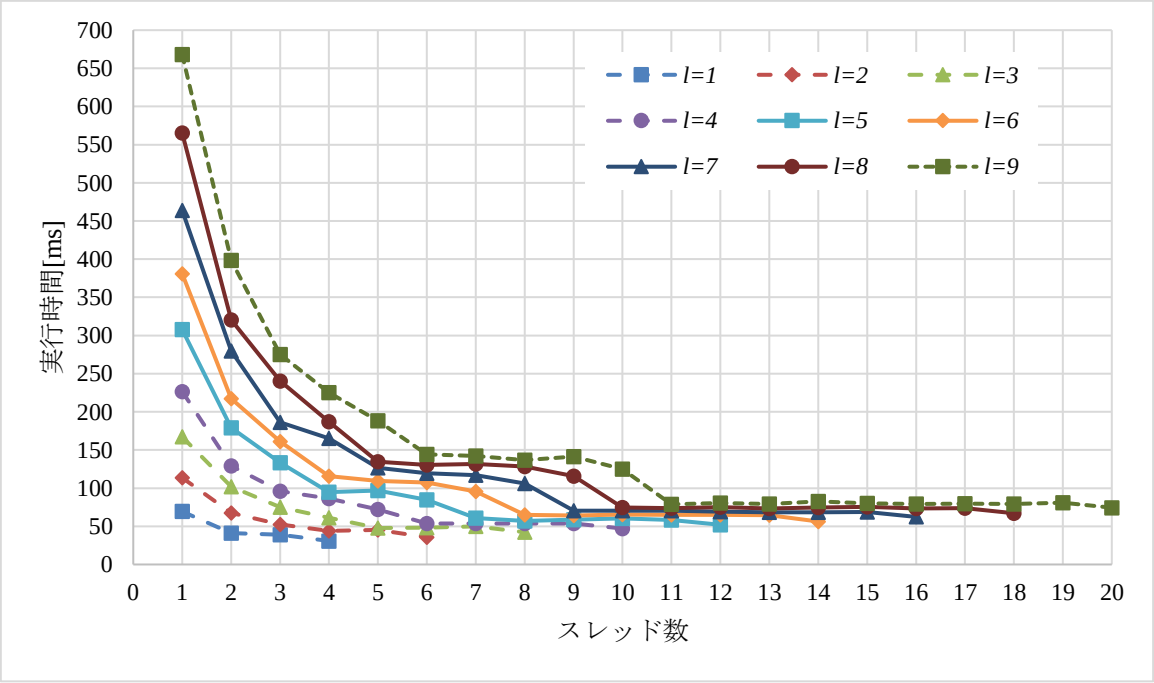

図 2.13　レベル*l*におけるスレッド数による *Relinearization* の実行時間の変化

表 2.5　*Rescaling* に対するマルチスレッドでの実行時間

スレッド数	レベルlでの実行時間[ms]								
	$l=1$	$l=2$	$l=3$	$l=4$	$l=5$	$l=6$	$l=7$	$l=8$	$l=9$
1	31.0	52.1	74.1	95.6	121.8	138.0	155.9	175.4	194.7
2	17.4	28.3	40.1	50.1	61.2	73.4	82.9	92.9	103.3
3	18.1	24.3	28.5	39.8	46.9	50.9	62.4	65.6	72.5
4	13.4	17.9	24.2	28.9	35.2	39.8	46.7	50.1	61.6
5		17.9	23.9	24.3	29.6	35.9	40.1	45.8	44.7
6		13.8	18.2	24.6	25.2	29.5	35.8	36.7	40.3
7			18.7	24.4	25.4	25.7	29.5	36.7	35.9
8			14.6	18.8	24.5	25.5	26.1	29.8	35.6
9				18.8	24.6	25.1	25.9	25.8	30.6
10				15.4	19.7	24.9	26.1	26.1	26.2
11					18.9	24.7	25.4	25.6	26.2
12					15.6	19.9	24.8	25.4	26.1
13						19.9	25.0	25.0	26.0
14						15.8	20.1	24.9	26.0
15							20.0	25.2	25.7
16							16.6	20.2	25.1
17								20.4	25.2
18								17.6	21.4
19									21.8
20									19.0

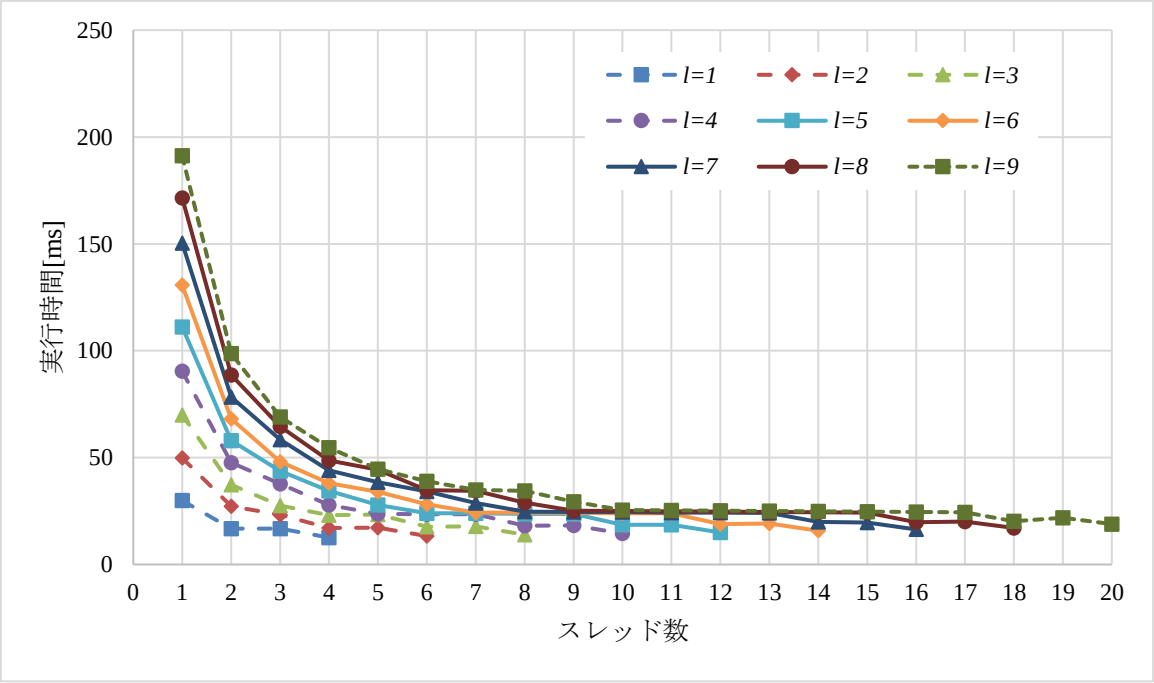

図 2.14　レベル*l*におけるスレッド数による *Rescaling* の実行時間の変化

第3章　　関連研究

本章では，関連研究を示す．

3.1 準同型暗号を用いたアプリケーションに対する最適化手法

完全準同型暗号を用いたアプリケーションでは，実行時間が特に長い準同型演算である *Bootstrapping* や *Relinearization* を実行する回数を減らすことで，ジョブのレイテンシを短縮することができる．そこで，「暗号文の値を条件に用いた条件分岐を行うことができない」という特徴を用いた静的な最適化について研究されている．2013 年に Lepoint らが，*Bootstrapping* の最適実行位置を決定する問題は NP 完全であると示すとともに，アプリケーションのフローグラフに対して，初期レベルを設定した場合における *Bootstrapping* の最小回数を求める手法を示した[20]．2017 年に佐藤らは，ループアンローリングを用いて for ループ内で実行される *Bootstrapping* の回数を減らす手法を提案した[9]．また，2019 年に佐藤らは，制約を付与することで，*Bootstrapping* や *Relinearization* の最小回数を近似解もしくは厳密解を実時間で求めることができることを示した[21]．同様に，完全準同型暗号の知識を持たないユーザであっても完全準同型暗号を用いたアプリケーションを利用できるようするコンパイラを開発することを目的とした研究も存在する[10–13]．具体的には，暗号文のパラメータを調整する準同型演算を自動で挿入する手法[12]，論理回路における AND ゲート，つまり準同型乗算の回数が少なくなるように回路を構成し，さらに実行時に準同型演算単位でリストスケジューリングを用いる手法[13]，準同型暗号上でのニューラルネットワークにおける推論処理を対象として，必要な準同型演算やレベルが少なくなるように準同型演算の実行順序を静的に最適化する手法[10], [11]が存在する．

しかし，いずれも静的な最適化か，準同型演算単位でのリストスケジューリングを用いた動的な並列化のみである．つまり，完全準同型暗号を用いたアプリケーションに対して，各クエリの到着時刻に依存して負荷が変化する状況に対応するために，1 つの準同型演算あたりの並列数を動的に決定する手法は，筆者らが SWoPP2019 に発表した手法[22]を除いて存在しない．

3.2 クライアント—サーバアプリケーションにおける高速化手法

本論文が対象とする，動的に計算資源を割り当てる既存研究も存在する．例えば， 2013 年に Anagnostopoulos ら及び 2018 年に Tsoutsouras らが発表した手法では，クライアント—サーバアプリケーションにおいて，ジョブの実行開始を管理する Init コアや，ジョブごとに用意された，各ジョブを管理する Manager コア，アイドル状態のコアを検出するための Controller コアを用いることで, 各ジョブに対して動的に CPU コアの割り当てを行う[2], [3]. 具体的には, 定期的に Manager コアが希望の CPU コア数を他の Manager コアや Controller コアに要求する．また，新たにジョブ

を実行開始する際には，Init コアが Controller コアや Manager コアに希望コア数を要求する．この時，割り当てるコア数の変動による実行速度の変化を基準に，要求を拒否するかどうかを決めている．また，2018 年に TsouTsouras らが発表した手法では，消費電力の削減も行っている[3].

2015 年に Singh らは，メニーコア CPU を搭載した複数のノードに対して，多数のジョブが到着する状況を対象に，ジョブに対してどのノードとコアを割り当てるかを動的に割り当てる手法を発表した[23]．ジョブごとに実行完了するまでの時間に応じて，価値を設定し，価値を最大化することが目的である．事前にジョブに割り当てるコア数による実行時間と価値を求めておき，新たにジョブが到着した時に計算資源が足りなければ，実行中ジョブの内，実行を中断しても最終的に得られる価値の合計に影響を与えにくいジョブを中断することで，新たなジョブのための計算資源を確保する．すなわち，実行しても得られる価値が少ないジョブを実行する優先度を下げ，かつジョブ単位で計算資源を割り当てる手法である．なお，ジョブの種類数は複数存在する．

3.3 タスク並列化の粒度を決定する手法

OSCAR コンパイラと呼ばれる，マルチグレイン並列化を行うコンパイラが存在する．与えられたアプリケーションコードに対して，マクロタスクレベルや演算レベルといった複数のレベルでの並列性を抽出することで，高い並列性による高速化を実現することができる．そして，得られた様々な粒度のタスクに対して，どのようにプロセッサを割り当てるか，という問題を解決する手法を 2003 年に小幡らが発表した[24]．しかし，単一ジョブに対して適用しているため，複数のジョブを同時に実行する状況を想定していない．

ハードリアルタイムアプリケーションでは，最悪実行時間を用いて，周期タスクをスケジューリングする手法が存在する．具体的には，自動車のエンジンコントロールユニットを対象とした静的なスケジューリングを行う研究[25]が存在する．しかし，周期タスクと異なり，各クエリの到着時刻に依存して負荷が変化する状況に対応するためには，静的な解析及び並列化では不十分である．そこで，実行時に動的にタスク分割やタスク統合を行うといった手法が研究されている．具体的には，一定期間ごとに，多数の実行待ちタスクをどのように組み合わせて，まとめて 1 つ CPU コアに割り当てるとデッドラインを守ることができるかを動的に求める手法が存在する[4]．また，まとめたタスクの実行時間が短い場合は，別のタスクを 2 つに分割した一方を追加し，他方を実行待ちタスクのキューへ追加するといった手法も存在する[26].

3.4 関連研究のまとめ

3.1〜3.3 節で示した関連研究を表 3.1 にまとめる．

表 3.1　関連研究の比較

アプリケーション種別	手法	静的な最適化手法	動的な最適化手法
準同型暗号を用いたアプリケーション	2013 年 Lepoint ら[20] 2017 年 佐藤ら[9] 2019 年 佐藤ら[21]	*Bootstrapping* や *Relinearization* といった実行時間が長い準同型演算の実行位置や実行回数の最適化	—
	2018 年 Crockett ら[12]	Relinearization といった暗号文のパラメータを調整する準同型演算を自動挿入	準同型演算単位の並列化
	2015 年 Carpov ら[13]	論理回路作成時に必要な AND ゲート（準同型乗算）の数を削減	準同型演算単位で並列実行
	2018 年 Boemer ら[10] 2019 年 Boemer ら[11]	ニューラルネットワークにおける演算に関して，必要な準同型演算の回数やレベルを削減	—
クライアント―サーバアプリケーション	2013 年 Anagnostopoulos ら[2] 2018 年 TsouTsouras ら[3]	—	ジョブ実行開始の管理や利用可能 CPU コアの管理，ジョブ内タスクの実行や利用 CPU コア数の管理をそれぞれ別々の CPU コアが実施
	2015 年　Singh ら[23]	—	実行中ジョブの中断やジョブに CPU コアを動的に割り当て
タスクの粒度の決定	2003 年　小幡ら[24]	複数粒度でタスクを分割	利用可能な計算資源に応じて，タスクの粒度を決定して実行

第4章　　提案手法　DAMCREM：Dynamic Allocation Method of Computation REsource to Macro-Tasks

本章では提案手法 DAMCREM について説明する．各クエリの到着時刻に依存して負荷が変化する状況において，平均レイテンシを短縮することが目的である．DAMCREM では，アプリケーションをマクロタスクグラフで表現する．マクロタスクは，実行順序のスケジューリングや計算資源を割り当てる対象の最小単位であり，1 つ以上の準同型演算で構成される．具体的には，まず，準同型乗算と *Relinearization*，*Rescaling* はそれぞれ独立したマクロタスクとする．その後，実行時間の短い準同型加算や *Simple Modulus Reduction* を既存のマクロタスクへ統合した．そして，マクロタスクごとに予め使用するスレッド数の候補を用意する．実行時に，実行中マクロタスクや実行待ちマクロタスクの種類や数に応じて，候補から選択したスレッド数でマクロタスクを実行する．

説明のために使用する記号について 4.1 節で示す．その後，4.2 節で問題をモデル化して解決すべき問題を明確にする．そして，4.3 節で明確にした解決すべき問題を解決する提案手法 DAMCREM の詳細を示す．

4.1 記号の定義

本章で用いる記号の定義を表 4.1 に示す．

表 4.1　提案手法の説明で使用する記号の定義（その 1）

記号	定義
P	● マクロタスクを実行するために使用できる総スレッド数
J_i	● ジョブ ID がiであるジョブ ● $1 \leq i < i'$の時，J_iのクエリ到着時刻は，$J_{i'}$のクエリ到着時刻よりも早い
$MT_{i,j}$	● ジョブ ID がi，マクロタスク ID がjであるマクロタスク ● マクロタスク ID jは，マクロタスクグラフを作成する際に，各マクロタスクに 1 から順に割り振られる． ● 本論文では，全てのジョブが同じ種類のアプリケーションを実行するため，$i \neq i'$に対して，$MT_{i,j}$と$MT_{i',j}$は，入出力となる平文や暗号文のパラメータ，実行する準同型演算は同じである．
$p_{i,j}$	● $MT_{i,j}$の優先度
$C_{i,j}$	● $MT_{i,j}$に割り当てるスレッド数の候補集合 ● $C_{i,j} = \left\{th_{i,j}^{(cand)}(1), th_{i,j}^{(cand)}(2), \cdots, 1\right\}$
$th_{i,j}^{(cand)}(k)$	● $MT_{i,j}$に割り当てるスレッド数のk番目の候補 ● $th_{i,j}^{(cand)}\left(\left\|C_{i,j}\right\|\right) = 1$ ● $P \geq th_{i,j}^{(cand)}(k) > th_{i,j}^{(cand)}(k'),\ \left(1 \leq k < k' \leq \left\|C_{i,j}\right\|\right)$
$r_{i,j}(n)$	● $MT_{i,j}$にnスレッド割り当てる優先度 ● $n \in C_{i,j}$ ● 4.3.4 項及び式 1 にて詳細を示す
$th_{i,j}^{(rel)}(i',j',n')$	● $r_{i',j'}(n')$を超える優先度を持つスレッド数の候補の内，最も多いスレッド数 ● $th_{i,j}^{(rel)}(i',j',n') = \underset{n}{\mathrm{argmax}}\left\{r_{i,j}(n) > r_{i',j'}(n') \middle\| n \in C_{i,j}\right\}$ ● $r_{i',j'} < r_{i,j}\left(th_{i,j}^{(rel)}(i',j',n')\right)$ ● $th_{i,j}^{(rel)}(i',j',n') \in C_{i,j}$
M	● ジョブに含まれる「マクロタスクの各使用スレッド数の優先度」の種類数 ➢ 常に 1 スレッドのみ割り当てられるマクロタスクは該当する全てのマクロタスクについて「マクロタスクの各使用スレッド数の優先度」が同じ種類とする ➢ 入力となる平文や暗号文のパラメータが同じかつ実行する準同型演算の種類や順序，使用スレッド数の候補が同じであれば，該当する全てのマクロタスクについて「マクロタスクの各使用スレッド数の優先度」が同じ種類とする

表 4.1　提案手法の説明で使用する記号の定義（その 2）

$W(t)$	● ある時刻tにおける，実行待ちマクロタスク数 ● データ依存が解決した未実行マクロタスクのみが対象である
$w_i^{(Job)}(t)$	● ある時刻tにおける，i番目の実行待ちマクロタスクのジョブ ID ➢ iが小さいほど優先度が高い ($i=1$が最も優先度が高い) ➢ $1 \leq i \leq W(t)$
$w_i^{(MT)}(t)$	● ある時刻tにおける, i番目の実行待ちマクロタスクのマクロタスク ID ➢ iが小さいほど優先度が高い ($i=1$が最も優先度が高い) ➢ $1 \leq i \leq W(t)$
$MT_{w_i^{(Job)}(t),w_i^{(MT)}(t)}$	● ある時刻tにおける，i番目の実行待ちマクロタスク ➢ iが小さいほど優先度が高い ($i=1$が最も優先度が高い) ➢ $MT_{w_1^{(Job)}(t),w_1^{(MT)}(t)}$は，ある時刻$t$での次に実行するマクロタスク ➢ $1 \leq i \leq W(t)$
$E(t)$	● ある時刻tにおける，実行中マクロタスクの数
$e_i^{(Job)}(t)$	● ある時刻tにおける，i番目の実行中マクロタスクのジョブ ID ➢ iが小さいほど実行開始時刻が早い. ➢ $1 \leq i \leq E(t)$
$e_i^{(MT)}(t)$	● ある時刻tにおける，i番目の実行中マクロタスクのマクロタスク ID ➢ iが小さいほど実行開始時刻が早い. ➢ $1 \leq i \leq E(t)$
$MT_{e_i^{(Job)}(t),e_i^{(MT)}(t)}$	● ある時刻tでのi番目の実行中マクロタスク ➢ iが小さいほど実行開始時刻が早い. ➢ $1 \leq i \leq E(t)$
$th^{(exec)}_{e_i^{(Job)}(t),e_i^{(MT)}(t)}$	● ある時刻tでの$MT_{e_i^{(Job)}(t),e_i^{(MT)}(t)}$に使用しているスレッド数 ● $1 \leq th^{(exec)}_{e_i^{(Job)}(t),e_i^{(MT)}(t)} \leq P^{(exec)}(t)$
$P^{(exec)}(t)$	● ある時刻tでマクロタスク実行に使用しているスレッド数の合計 ● $0 \leq P^{(exec)}(t) = \sum_{i=1}^{E(t)} th^{(exec)}_{e_i^{(Job)}(t),e_i^{(MT)}(t)} \leq P$
$P^{(rest)}(t)$	● ある時刻tでマクロタスク実行に使用可能なスレッド数の合計 ● $0 \leq P^{(rest)}(t) \leq P - P^{(exec)}(t)$

4.2 問題のモデル化

本節では，問題をモデル化し，解決すべき問題を明確にする．実行待ちマクロタスクを実行開始する過程を用いて説明する．

(1) 実行待ちマクロタスクの実行順序のスケジューリング

まず，ある時刻tにおいて，図 4.1 に示すように，$E(t)$個の実行中マクロタスクと$W(t)$個の

実行待ちマクロタスクで構成されたマクロタスクリストが存在する．そして，次に実行するマクロタスクを実行待ちマクロタスクリストから 1 つ選択する．ここでは，実行待ちマクロタスクリストの内，先頭にあるマクロタスクを優先して選択するとする．この時，1 つの目の問題として，「実行待ちマクロタスクリストにおけるマクロタスクをどのような順番で並べておくべきか」という問題がある．

$E(t)$個の実行中マクロタスクリスト			
マクロタスク	$MT_{e_1^{(Job)}(t),e_1^{(MT)}(t)}$	…	$MT_{e_{E(t)}^{(Job)}(t),e_{E(t)}^{(MT)}(t)}$
使用スレッド数	$th^{(exec)}_{e_1^{(Job)}(t),e_1^{(MT)}(t)}$	…	$th^{(exec)}_{e_{E(t)}^{(Job)}(t),e_{E(t)}^{(MT)}(t)}$

$W(t)$個の実行待ちマクロタスクリスト			
マクロタスク	$MT_{w_1^{(Job)}(t),w_1^{(MT)}(t)}$	…	$MT_{w_{W(t)}^{(Job)}(t),w_{W(t)}^{(MT)}(t)}$
使用スレッド数の候補	$C_{w_1^{(Job)}(t),w_1^{(MT)}(t)}$	…	$C_{w_{W(t)}^{(Job)}(t),w_{W(t)}^{(MT)}(t)}$

図 4.1　ある時刻tでの，実行中マクロタスクと実行待ちマクロタスクの状態

(2) 次に実行するマクロタスク$MT_{w_1^{(Job)}(t),w_1^{(MT)}(t)}$に使用するスレッド数の決定

その後，図 4.2 に示すように，次に実行するマクロタスク$MT_{w_1^{(Job)}(t),w_1^{(MT)}(t)}$に対して，使用するスレッド数$th^{(exec)}_{e_{E(t)+1}^{(Job)}(t),e_{E(t)+1}^{(MT)}(t)}$を決定する．ここで，2 つ目の問題として，「各マクロタスクに対して使用するスレッド数の候補をどのように静的に設定しておくか」という問題がある．さらに，3 つ目の問題点として，「使用するスレッド数の候補の内，どのスレッド数を動的に選択するべきか」という問題がある．

$E(t)$個の実行中マクロタスク+1個の実行開始マクロタスクリスト				
マクロタスク	$MT_{e_1^{(Job)}(t),e_1^{(MT)}(t)}$	…	$MT_{e_{E(t)}^{(Job)}(t),e_{E(t)}^{(MT)}(t)}$	$MT_{e_{E(t)+1}^{(Job)}(t),e_{E(t)+1}^{(MT)}(t)}$
使用スレッド数	$th^{(exec)}_{e_1^{(Job)}(t),e_1^{(MT)}(t)}$	…	$th^{(exec)}_{e_{E(t)}^{(Job)}(t),e_{E(t)}^{(MT)}(t)}$	$th^{(exec)}_{e_{E(t)+1}^{(Job)}(t),e_{E(t)+1}^{(MT)}(t)}$

- $th^{(exec)}_{e_{E(t)+1}^{(Job)}(t),e_{E(t)+1}^{(MT)}(t)} \in C_{w_1^{(Job)}(t),w_1^{(MT)}(t)}$

$W(t)$個の実行待ちマクロタスクリスト			
マクロタスク	$MT_{w_1^{(Job)}(t),w_1^{(MT)}(t)}$	…	$MT_{w_{W(t)}^{(Job)}(t),w_{W(t)}^{(MT)}(t)}$
使用スレッド数の候補	$C_{w_1^{(Job)}(t),w_1^{(MT)}(t)}$	…	$C_{w_{W(t)}^{(Job)}(t),w_{W(t)}^{(MT)}(t)}$

図 4.2　ある時刻tでの，次に実行するマクロタスクに対する使用スレッド数決定直後の状態

(3) 新たにマクロタスク$MT_{w_1^{(Job)}(t),w_1^{(MT)}(t)}$を実行した直後の処理

次に実行するマクロタスク$MT_{w_1^{(Job)}(t),w_1^{(MT)}(t)}$とマクロタスク$MT_{w_1^{(Job)}(t),w_1^{(MT)}(t)}$に使用するスレッド数$th^{(exec)}_{e_{E(t)+1}^{(Job)}(t),e_{E(t)+1}^{(MT)}(t)}$を決定し，マクロタスク$MT_{w_1^{(Job)}(t),w_1^{(MT)}(t)}$を実行開始した直後の時刻$t' > t$における状態を図 4.3 に示す．その後，(1) の状態に戻り処理を行うことを繰り返す．ただし，(1) の処理に戻る前に，実行中マクロタスクのいずれかが実行完了するか，新たなジョブが実行開始する可能性がある．この場合，実行待ちマクロタスクリストに新たなマクロタスクが追加され，実行待ちマクロタスクリストが更新されてから (1) の処理を

実施する．

$E(t')(=E(t)+1)$個の実行中マクロタスクリスト				
マクロタスク	$MT_{e_1^{(Job)}(t'),e_1^{(MT)}(t')}$	…	$MT_{e_{E(t')-1}^{(Job)}(t'),e_{E(t')-1}^{(MT)}(t')}$	$MT_{e_{E(t')}^{(Job)}(t),e_{E(t')}^{(MT)}(t')}$
使用スレッド数	$th^{(exec)}_{e_1^{(Job)}(t'),e_1^{(MT)}(t')}$	…	$th^{(exec)}_{e_{E(t')-1}^{(Job)}(t'),e_{E(t')-1}^{(MT)}(t')}$	$th^{(exec)}_{e_{E(t')}^{(Job)}(t'),e_{E(t')}^{(MT)}(t')}$

- $e_i^{(Job)}(t') = e_i^{(Job)}(t),\ \left(1 \le i < E(t')\right)$
- $e_i^{(MT)}(t') = e_i^{(MT)}(t),\ \left(1 \le i < E(t')\right)$
- $e_{E(t')}^{(Job)}(t') = w_1^{(Job)}(t)$
- $e_{E(t')}^{(MT)}(t') = w_1^{(MT)}(t)$
- $w_i^{(Job)}(t') = w_{i+1}^{(Job)}(t),\ \left(1 \le i < W(t')\right)$
- $w_i^{(MT)}(t') = w_{i+1}^{(MT)}(t),\ \left(1 \le i < W(t')\right)$

$W(t')(=W(t)-1)$個の実行待ちマクロタスクリスト			
マクロタスク	$MT_{w_1^{(Job)}(t'),w_1^{(MT)}(t')}$	…	$MT_{w_{W(t')}^{(Job)}(t'),w_{W(t')}^{(MT)}(t')}$
使用スレッド数の候補	$C_{w_1^{(Job)}(t'),w_1^{(MT)}(t')}$	…	$C_{w_{W(t')}^{(Job)}(t'),w_{W(t')}^{(MT)}(t')}$

図 4.3　ある時刻tでの次のマクロタスクを実行した直後の時刻t'の状態

4.3 提案手法 DAMCREM の詳細

本節では，4.2 節で示した 3 つの問題を解決するために用いる手法を示す．DAMCREM の説明に，図 4.4 に示すマクロタスクグラフを使用する．アプリケーションが与えられた時に，マクロタスクグラフを作成する方法は，4.3.1 項で示す．DAMCREM では，全てのジョブが同一アプリケーションを実行するとし，クエリとなる暗号文のパラメータは固定とする．したがって，異なる 2 つのジョブ ID $i \neq i'$と任意のマクロタスク ID jについて，マクロタスク$MT_{i,j}$とマクロタスク$MT_{i',j}$は，入力データは異なるが，実施する準同型演算の種類や順番は同じである．使用するマクロタスクグラフは DAG(Direct Acyclic Graph)で表されている．ジョブ開始時は，$\{MT_{i,1}, MT_{i,2}, MT_{i,3}\}$の 3 つのマクロタスクのみが実行可能である．図中に記載した，$p_{i,j}$を導出する方法は 4.3.2 項の **(2)** に示す．

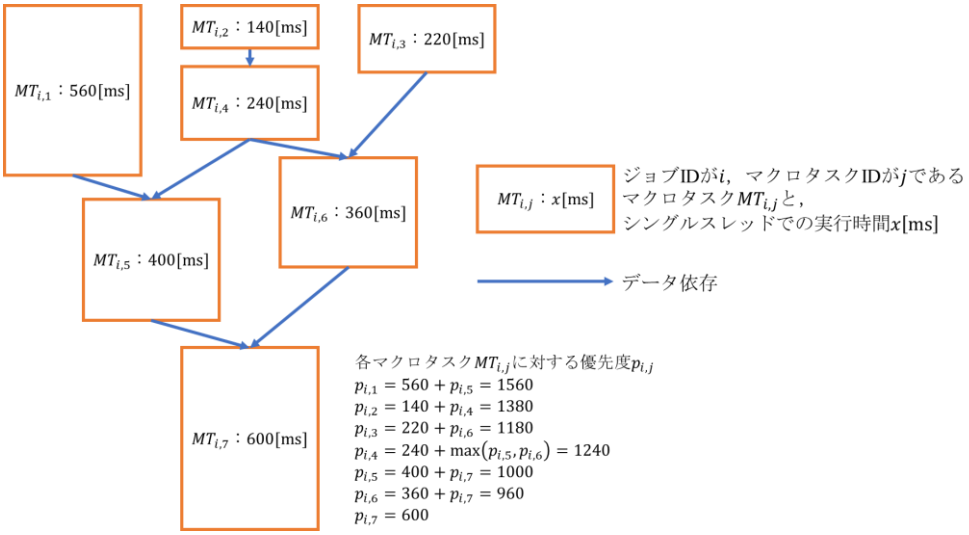

図 4.4　提案手法 DAMCREM の説明に使用するマクロタスクグラフ（ジョブ ID をiとする）

4.3.1　マクロタスク

本項では，DAMCREM で用いるマクロタスクとマクロタスクグラフについて説明する．

まず，DAMCREM におけるマクロタスクの定義について説明する．マクロタスクを，実行順序の決定や計算資源の割り当ての対象となる最小単位とする．マクロタスクは，1 つ以上の準同型演算で構成される．1 つのマクロタスクは，準同型乗算か *Relinearization*，*Rescaling* のいずれか 1 つが含まれる．さらに，1 つのマクロタスクの粒度を大きくして，マクロタスク数増加によるオーバヘッドを削減することを目的として，実行時間の短い準同型加算や *Simple Modulus Reduction* は，他のマクロタスクに統合する．つまり，1 つのマクロタスクには，準同型乗算か *Relinearization*，*Rescaling* のいずれか 1 つと，0 個以上の準同型加算や *Simple Modulus Reduction* で構成される．

アプリケーションが与えられた時に，マクロタスクグラフを作成する方法について説明する．下記の手順でマクロタスクグラフを手動で作成する．下記に示す操作 6 及び 7 の擬似コードをアルゴリズム 4.1 に示す．

1. 定数である平文もしくは暗号文を入力とする *Simple Modulus Reduction* はジョブ開始前に適用し，マクロタスクグラフには含めない．

2. 必要なレベルが少なくなるように準同型乗算の実行順序を決定する．例えば，$\mathrm{Enc}(\tilde{x})\otimes\mathrm{Enc}(\tilde{x})\otimes\mathrm{Enc}(\tilde{x})\otimes\mathrm{Enc}(\tilde{x})$を計算する場合は，$\big(\mathrm{Enc}(\tilde{x})\otimes\mathrm{Enc}(\tilde{x})\big)\otimes\big(\mathrm{Enc}(\tilde{x})\otimes\mathrm{Enc}(\tilde{x})\big)$とすることで，2 乗を計算する準同型乗算と 4 乗を計算する準同型乗算の計 2 回の準同型乗算が必要であり，必要なレベルは 2 である．一方，$\Big(\big(\mathrm{Enc}(\tilde{x})\otimes\mathrm{Enc}(\tilde{x})\big)\otimes\mathrm{Enc}(\tilde{x})\Big)\otimes\mathrm{Enc}(\tilde{x})$とすると，必要な準同型乗算は 3 回であり，必要なレベルは 3 である．必要なレベルが増加することで，2.3.2 項で示した通り，実行時間が長くなる．

3. *Relinearization* と *Rescaling* の実行回数が少なくなるように，準同型演算の実行順序を決定する．例えば，2.3.2 項で示したように，準同型乗算後に準同型加算を行う場合，準同型加算の入力暗号文のパラメータが同じであれば準同型加算を *Relinearization* や *Rescaling* よりも前に適用するようにする．ただし，準同型乗算の出力暗号文を入力とする準同型加算や準同型乗算が複数個続く場合のように，アプリケーション全体での *Relinearization* や *Rescaling* の実行回数が変化しないもしくは増加してしまう場合は，後続の準同型加算や準同型乗算より先に *Relinearization* や *Rescaling* を適用する．
   理由を説明する．長さS，レベルlの暗号文に対して，準同型加算や *Relinearization*，*Rescaling* を適用した時の実行時間をそれぞれ$T_{add}(S,l)$，$T_{relin}(S,l)$，$T_{rescl}(S,l)$とする．2 つの暗号文を入力とする準同型乗算$\mathrm{Enc}(\tilde{x}_1)\otimes\mathrm{Enc}(\tilde{y}_1)\rightarrow\mathrm{Enc}(\tilde{z}_1)$及び$\mathrm{Enc}(\tilde{x}_2)\otimes\mathrm{Enc}(\tilde{y}_2)\rightarrow\mathrm{Enc}(\tilde{z}_2)$の後に実行される準同型加算と *Relinearization*, *Rescaling* の実行順序を変えた時の実行時間を示す．下記説明では，$\mathrm{Size}\big(\mathrm{Enc}(\tilde{z}_1)\big)=\mathrm{Size}\big(\mathrm{Enc}(\tilde{z}_2)\big)=3$，$\mathrm{Level}\big(Enc(\tilde{z}_1)\big)=\mathrm{Level}\big(Enc(\tilde{z}_2)\big)=l'$とする．
   （1）*Relinearization*，*Rescaling*，準同型加算の順で実行した場合：
   合計実行時間は，*Relinearization* と *Rescaling* が 2 回ずつ，準同型加算が 1 回であるため，$2T_{relin}(3,l')+2T_{rescl}(2,l')+T_{add}(2,l'-1)$である．
   （2）準同型加算，*Relinearization*，*Rescaling* の順で実行した場合：
   合計実行時間は，準同型加算と *Relinearization*，*Rescaling* がそれぞれ 1 回ずつであるため，$T_{add}(3,l')+T_{relin}(3,l')+T_{rescl}(2,l')$である．
   （1）と（2）の差は，
   $$\big(2T_{relin}(3,l')+2T_{rescl}(2,l')+T_{add}(2,l'-1)\big)-\big(T_{add}(3,l')+T_{relin}(3,l')+T_{rescl}(2,l')\big)$$
   $$=T_{relin}(3,l')+T_{rescl}(2,l')+T_{add}(2,l'-1)-T_{add}(3,l')$$
   である．準同型加算の時間計算量は，長さとレベルのそれぞれに対して線形に比例する．したがって，2.3.2 項で示したように，$T_{relin}(3,l')+T_{rescl}(2,l')>T_{add}(3,l')-T_{add}(2,l'-1)$であるため，（2）の方が実行時間を短くすることができる．

4. 複数の平文及び暗号文の総和を計算する場合は，各準同型加算の出力暗号文を同じデータに上書きする．具体的には，入力暗号文$a_1,\cdots,a_m$，総和結果を格納する暗号文をxとする場合，暗号文xの初期値をa_1として，$x=x+a_i$を$2\le i\le m$に対して行う．

5. 2～4 の操作によって得られた準同型演算の実行順序とデータ依存関係を DAG で表す．

6. 準同型乗算と *Relinearization*，*Rescaling* をそれぞれ 1 つの独立したマクロタスクとする．

7. 実行時間が短い準同型加算及び *Simple Modulus Reduction* を，それぞれの準同型演算の直前の準同型演算f'が 1 つのみであり，その準同型演算f'の出力を使用する他の準同型演算が存在しなければ，f'が属するマクロタスクに統合する．
   f'が 1 つも存在しないもしくは 2 つ存在するか，f'の出力を使用する他の準同型演算が存在するのであれば，それぞれの準同型演算の出力暗号文を入力とするマクロタスク$MT_{i,j}$に統合する．ただし，統合するマクロタスクが存在しない場合や，一意に決まらない場合は，

下記に示した方法で統合する．

i. アプリケーションの最後に実行される準同型加算は，対象の準同型加算の出力暗号文を入力とするマクロタスクが存在しないため，対象の準同型加算の入力暗号文を出力するマクロタスク$MT_{i,j}$に統合する．ただし，マクロタスク$MT_{i,j}$が複数存在する場合は，クリティカルパスが長くなるのを回避するために，独立したマクロタスクとする．

ii. 条件を満たすマクロタスクMT_{ij}が複数存在する場合は，マクロタスク$MT_{i,j}$のそれぞれにおいて，対象の準同型加算もしくは *Simple Modulus Reduction* を実行する．

マクロタスク間のデータ依存の内，出力依存を解消するために，あるマクロタスク$MT_{i,j}$に対して，順依存のマクロタスク$MT_{i,j'}$が複数存在する場合は，マクロタスク$MT_{i,j'}$のそれぞれは，マクロタスク$MT_{i,j}$の出力暗号文をコピーして入力暗号文として使用する．マクロタスク$MT_{i,j'}$が 1 つのみの場合は，マクロタスク$MT_{i,j}$の出力暗号文をコピーせずにマクロタスク$MT_{i,j'}$の入力暗号文として使用する．

アルゴリズム 4.1　マクロタスクグラフ作成における操作 6 及び 7 の擬似コード

	入力：上記操作 5 で得られたアプリケーションの DAG 出力：マクロタスクグラフを構成するマクロタスクの集合V
1	function add_homfunc_to_macrotask(f, F)　// 準同型演算fをマクロタスクに統合するための補助関数．Fはfと同じマクロタスクに含まれることになる準同型演算の集合
2	if (fが既にマクロタスクMTに属している)
3	MTにFに属する各準同型演算を追加
4	else if (f及びFの各準同型演算への入力を出力する準同型演算$f' \notin F$が 1 つのみ存在する)
5	add_homfunc_to_macrotask($f', \{f\} \cup F$)
6	else if (fの出力を入力とする準同型演算が存在しない)
7	Vにマクロタスク$\{f\} \cup F$を追加
8	else
9	for f'' in {fの出力を用いる準同型演算}
10	if (f''を含むマクロタスクが存在しない)
11	add_homfunc_to_macrotask ($f'', \{f\}$)
12	else
13	f''を含むマクロタスクにfを追加
14	
15	function make_macrotask_graph()　// マクロタスクグラフを作成する関数
16	$V \leftarrow \{\}$
17	for f in {DAG に存在する準同型乗算及び$Relinearization$，$Rescaling$}
18	Vにマクロタスク$\{f\}$を追加
19	for f in {DAG に存在する準同型加算や$Simple\ Modulus\ Reduction$}
20	add_homfunc_to_macrotask($f, \{\}$)

4.3.2　実行待ちマクロタスクリストにおけるマクロタスクの順序

本項では，1 つ目の問題である「実行待ちマクロタスクリストにおけるマクロタスクをどのような順番で並べておくべきか」という問題を解決する手法を示す．ジョブの平均レイテンシを短縮するために，クエリ到着時刻が早いジョブを優先し，その上で，データ依存が解決した，クリティカルパスが長いマクロタスクを優先して実行する．実行時のオーバヘッドを削減するために，新たなマクロタスクが実行待ちマクロタスクリストに追加されるたびにマクロタスクの順序を決定する．異なるジョブに属するマクロタスク間の優先度と同一ジョブに属するマクロタスク間での優先度の 2 つの観点で順序を決定する．

(1) 異なるジョブに属するマクロタスク間の優先度

まず，異なるジョブに属するマクロタスク間での優先度について説明する．DAMCREM では，クエリの到着が早いジョブのマクロタスクの優先度を高くする．具体的には，$i < i'$とした時，全てのj, j'に対して$MT_{i,j}$の優先度は，$MT_{i',j'}$の優先度よりも高いとする．

理由を説明する．他に考えられる優先度の設定方法としては，ジョブを構成するマクロタスクごとに予め設定しておいたデッドラインを用いる方法である．しかし，本論文のゴールは，ジョブの平均レイテンシの短縮である．ジョブを構成するマクロタスクの内，最後に実行されるマクロタスク$MT_{i,last}$より前の全てのマクロタスクがデッドラインよりも十分早く実行完了したと仮定する．この時，マクロタスク$MT_{i,last}$を実行することで，i番目のジョブは実行完了となる．しかし，i番目のジョブのデッドラインまでの時間が十分に長いため，マクロタスク$MT_{i,last}$の優先度は低くなる．つまり，マクロタスク$MT_{i,last}$の実行が遅れるため，i番目のジョブのレイテンシを十分に短縮することができない．したがって，DAMCREM では，クエリの到着順を元に異なるジョブに属するマクロタスク間の優先度を設定した．

(2) 同一ジョブに属するマクロタスク間の優先度

次に，同一ジョブに属するマクロタスク間の優先度について説明する．固定優先度のリストスケジューリングを用いた．各マクロタスクに対して，シングルスレッド時の実行時間を元に優先度を設定する．あるマクロタスク$MT_{i,j}$の優先度は，そのクリティカルパス上にある全てのマクロタスクをシングルスレッドで実行した時の実行時間の総和とする．図 4.4 を用いて具体的に説明する．例えば，$MT_{i,4}$に対しては，「$MT_{i,4} \rightarrow MT_{i,5} \rightarrow MT_{i,7}$」と「$MT_{i,4} \rightarrow MT_{i,6} \rightarrow MT_{i,7}$」の 2 つのパスが存在する．「$MT_{i,5} \rightarrow MT_{i,7}$」の優先度は$p_{i,5} = 400 + p_{i,7} = 400 + 600 = 1000$である．一方，「$MT_{i,6} \rightarrow MT_{i,7}$」の優先度は$p_{i,6} = 360 + p_{i,7} = 360 + 600 = 960$である．つまり，「$MT_{i,4} \rightarrow MT_{i,5} \rightarrow MT_{i,7}$」のパスが「$MT_{i,4} \rightarrow MT_{i,6} \rightarrow MT_{i,7}$」のパスよりも実行時間が長い．よって，$MT_{i,4}$の優先度は$p_{i,4} = 240 + \max(p_{i,5}, p_{i,6}) = 240 + \max(1000, 960) = 240 + 1000 = 1240$となる．

あるマクロタスクについて，シングルスレッド時の実行時間をコストとして用いた理由を説明する．提案手法では，負荷に応じて動的に並列化を行うため，並列化すべきかどうかの判定によるオーバヘッドが生じる．マルチスレッド時の実行時間をコストとして用いると，負荷が大きい状況で不適切な順番でマクロタスクが実行される可能性がある．具体的には，クリティカルパス上にあるマクロタスクのコストが小さく見積もられ，優先度が低くなるた

めに，実行開始が遅れてしまうことが挙げられる．したがって，各マクロタスクのコストとしてシングルスレッド時の実行時間を採用した．

4.3.3 各マクロタスクに対する使用スレッド数の候補の静的設定

本項では，2 つ目の問題である「各マクロタスクに対して使用するスレッド数の候補をどのように静的に設定しておくか」という問題を解決する手法を示す．スレッド数の候補から動的に選択する時のオーバヘッドを削減するために，マクロタスクの実行時間を短縮する効果があるスレッド数を候補に選択する．

完全準同型暗号を用いたアプリケーションでは，暗号化された値を条件とした条件分岐ができない．したがって，本論文ではマクロタスクグラフの形状は静的に決定し，動的に変化しないとしている．よって，ジョブへのクエリである暗号文のパラメータが分かった段階で，マクロタスクで用いられる平文や暗号文のパラメータを知ることができる．そして，平文及び暗号文のパラメータが決まれば，各マクロタスクに使用するスレッド数の候補数は有限個である．しかし，暗号文のレベルが大きくなるほど，各マクロタスクに使用するスレッド数の候補数は増加する．提案手法では各マクロタスクに使用するスレッド数を動的に候補から選択するため，使用するスレッド数の候補数が多いほど，実行時のオーバヘッドが大きくなり，並列化による高速化の効果が減少してしまう．したがって，実行時のオーバヘッドを小さくする必要がある．

上記に示した通り，各マクロタスクに使用するスレッド数の候補数は有限個である．したがって，事前に各マクロタスクに対して，全ての使用スレッド数の候補で実行しておくことで，各候補を用いた時の実行時間を知ることができる．ここで，2.3.3 項で示した内容と同様に，スレッド数を増やした時に実行時間が短くなるような適切なスレッド数のみを候補として選ぶことで，実行時のオーバヘッドを小さくすることができる．そして，選んだ候補のスレッド数での各実行時間を 4.3.4 項で使用する．

4.3.4 次に実行するマクロタスクへの使用スレッド数の動的選択

本項では，3 つ目の問題である「使用するスレッド数の候補の内，どのスレッド数を動的に選択するべきか」という問題を解決する手法を示す．複数のスレッドを用いることによる高速化の効果が大きいマクロタスクに多くのスレッドが割り当てられるように，各マクロタスクに対するスレッド数の各候補に設定した優先度を用いて，次に実行するマクロタスクへの使用スレッド数を選択する．具体的には，下記に示す静的な処理と動的な処理に分けられる．

1. マクロタスクごとに，使用するスレッド数の各候補に優先度を静的に設定する
2. 下記 2 つの値を用いて，新たに実行するマクロタスクに使用するスレッド数を動的に決定

する

- 利用可能な総スレッド数$P^{(rest)}(t)$
- 各実行待ちマクロタスクのスレッド数の各候補の優先度

(1) マクロタスクごとに，使用するスレッド数の各候補に優先度を静的に設定する

具体例を用いて説明する．2 種類のマクロタスク（以下，マクロタスク 1 とマクロタスク 2）が存在し，使用スレッド数と実行時間の関係が表 4.2 に示す値とする．ただし，「—」は，該当のスレッド数が候補に含まれていないことを示す．ここで，式 1 を用いて，ジョブ ID がi，マクロタスク ID がjであるマクロタスク$MT_{i,j}$に対してnスレッド使用することに対する優先度$r_{i,j}(n)$を定義する．式 1 において，$T(n')$は対象のマクロタスクをn'スレッド使用した時の実行時間である．優先度の値は「対象のマクロタスク$MT_{i,j}$に対してnスレッド使用した時における，追加した 1 スレッドあたりの実行時間短縮率」である．

$$r_{i,j}(n) = \begin{cases} 1 & (n = 1) \\ \dfrac{T(1) - T(n)}{T(1)} \times \dfrac{1}{n-1} & (n > 1) \end{cases} \tag{1}$$

表 4.2 の各スレッド数について求めた優先度を表 4.3 に示す．優先度について具体的に説明する．

表 4.2　各マクロタスクの使用スレッド数nと実行時間の関係

マクロタスク ID	スレッド数nでの実行時間[ms]					
	$n=1$	$n=2$	$n=3$	$n=4$	$n=5$	$n=6$
1	220	120	100	80	—	70
2	560	340	—	240	—	140

※「—」はスレッド数nが候補に含まれていないことを示す．

表 4.3　各マクロタスクにnスレッド使用することの優先度

マクロタスクの番号	nスレッド使用することに対する優先度					
	$n=1$	$n=2$	$n=3$	$n=4$	$n=5$	$n=6$
1	1.000	0.455	0.273	0.212	—	0.136
2	1.000	0.393	—	0.190	—	0.150

※「—」はスレッド数nが候補に含まれていないために，優先度が存在しないことを示す．

次に実行するマクロタスクがマクロタスク 1 である場合

マクロタスク 1 に 4 スレッド使用する場合は，マクロタスク 2 には優先度が 0.212 を超える候補の内，最多スレッド数である 2 スレッドを使用する．また，他のジョブのマクロタスク 1 に対しては，優先度が 0.212 を超える候補の内，最多スレッド数である 3 スレッドを使用する．

次に実行するマクロタスクがマクロタスク 2 である場合

マクロタスク 2 に 2 スレッド使用する場合は，マクロタスク 1 には優先度が 0.393 を超える候補の内，最多スレッド数である 2 スレッドを使用する．また，他のジョブのマクロタスク 2 に対しては，優先度が 0.393 を超える候補の内，最多スレッド数である 1 スレッドを使用する．

同様にして，任意の 2 つのマクロタスクに対して使用するスレッド数の関係を静的に求めた結果を表 4.4 に示す．

表 4.4　表 4.3 で示したマクロタスク間の使用スレッド数の関係

次に実行するマクロタスクのマクロタスク ID $w_1^{(MT)}(t)$	使用するスレッド数n	2 番目以降の実行待ちマクロタスクのマクロタスク ID $w_i^{(MT)}(t), 2 \le i \le W(t)$と使用するスレッド数	
		1	2
1	1	1	1
	2	1	1
	3	2	2
	4	3	2
	6	4	4
2	1	1	1
	2	2	1
	4	4	2
	6	4	4

（2）新たに実行するマクロタスクに使用するスレッド数を動的に決定する

実行待ちマクロタスクの数$W(t) = 4$の場合を例として説明する．各マクロタスクに対する使用スレッド数と実行時間の関係を表 4.5 に示した値とする．「―」は，該当のスレッド数が候補に含まれていないことを示す．また，マクロタスク ID jが同じであれば，入力出力暗号文のパラメータや演算内容は同じである．そして，表 4.6 に各マクロタスクに対する優先度を示し，ジョブ内のマクロタスクの番号ごとに，優先度から求めたスレッド数の関係を表 4.7 に示す．さらに，表 4.6 で示した情報と表 4.7 で示した関係を用いて作成した，実行待ちマクロタスクリストにおける先頭のマクロタスク$MT_{w_1^{(Job)}(t),w_1^{(MT)}(t)}$とそれ以外のマクロタスク$MT_{w_i^{(Job)}(t),w_i^{(MT)}(t)}$(ただし，$1 < i \le W(t)$)とのスレッド数の関係を表 4.8 に示す. 最後に, 式 2 で示した条件を満たす最大のnを求め，マクロタスク$MT_{w_1^{(Job)}(t),w_1^{(MT)}(t)}$を$n$スレッドで実行する．

$$n + \sum_{i=2}^{W(t)} th^{(rel)}_{w_i^{(Job)}(t),w_i^{(MT)}(t)}\left(w_1^{(Job)}(t), w_1^{(MT)}(t), n\right) \le P^{(rest)}(t) \quad (2)$$

表 4.5　各マクロタスクに対する使用スレッド数と実行時間の関係

実行待ちマクロタスクの順番i	実行待ちマクロタスクのジョブ ID $w_i^{(Job)}(t)$	実行待ちマクロタスクのマクロタスク ID $w_i^{(MT)}(t)$	$n \in C_{w_i^{(Job)}(t),w_i^{(MT)}(t)}$スレッドでの実行時間[ms]					
			$n=1$	$n=2$	$n=3$	$n=4$	$n=5$	$n=6$
1	1	3	220	120	100	80	—	70
2	2	1	560	340	—	240	—	140
3	2	2	140	100	90	—	70	60
4	2	3	220	120	100	80	—	70

※「—」はスレッド数nが候補に含まれていないことを示す.

表 4.6　各マクロタスクにおける各使用スレッド数に対する優先度

実行待ちマクロタスクの順番i	実行待ちマクロタスクのジョブ ID $w_i^{(Job)}(t)$	実行待ちマクロタスクのマクロタスク ID $w_i^{(MT)}(t)$	nスレッド使用することの優先度$r_{w_i^{(Job)}(t),w_i^{(MT)}(t)}(n)$					
			$n=1$	$n=2$	$n=3$	$n=4$	$n=5$	$n=6$
1	1	3	1.000	0.455	0.273	0.212	—	0.136
2	2	1	1.000	0.393	—	0.190	—	0.150
3	2	2	1.000	0.286	0.179	—	0.125	0.114
4	2	3	1.000	0.455	0.273	0.212	—	0.136

※「—」はスレッド数nが候補に含まれていないために，優先度が存在しないことを示す.

表 4.7　表 4.6 を用いて算出した，$r_{i'j'}(n')$を超える優先度を持つ
マクロタスク$MT_{i,j}$の最多スレッド数$th_{i,j}^{(rel)}(i',j',n')$

マクロタスク ID j'	使用するスレッド数n'	他のマクロタスク$MT_{i,j}$における スレッド数$th_{i,j}^{(rel)}(i',j',n')$		
		$j=1$	$j=2$	$j=3$
1	1	1	1	1
	2	1	1	1
	3	2	2	2
	4	3	2	2
	6	4	4	3
2	1	1	1	1
	2	2	1	1
	4	4	2	2
	6	4	4	3
3	1	1	1	1
	2	2	2	1
	3	4	4	2
	5	6	6	3
	6	6	6	5

表 4.8　表 4.6 で示したマクロタスク$MT_{w_1^{(Job)}(t),w_1^{(MT)}(t)}$に使用するスレッド数に対する
他のマクロタスクに使用するスレッド数の関係

実行待ちマクロタスクの順番i	実行待ちマクロタスクのジョブ ID $w_i^{(Job)}(t)$	実行待ちマクロタスクのマクロタスク ID $w_i^{(MT)}(t)$	マクロタスク$MT_{w_1^{(Job)}(t),w_1^{(MT)}(t)}$に$n$スレッド使用する優先度$r_{w_1^{(Job)}(t),w_1^{(MT)}(t)}(n)$を超える優先度を持つ候補の内，最多スレッド数 $th^{(rel)}_{w_i^{(Job)}(t),w_i^{(MT)}(t)}\left(w_1^{(Job)}(t),w_1^{(MT)}(t),n\right)$				
			$n=1$	$n=2$	$n=3$	$n=4$	$n=6$
2	1	2	1	1	2	2	4
3	1	3	1	1	2	2	3
4	2	1	1	1	2	3	4

DAMCREM におけるスレッド数の動的選択に必要な最悪計算量を説明する．なお，使用スレッド数の動的選択においては，計算量削減のために，各使用スレッド数の優先度が同じマクロタスクを「『マクロタスクの各使用スレッド数の優先度』が同じ種類であるマクロタスク」として扱う．具体的には，2 つのマクロタスク（マクロタスク 1，マクロタスク 2）に対して，下記の 2 つ条件のどちらかもしくは両方を満たす場合は，マクロタスク 1 とマクロタスク 2 を「『マクロタスクの各使用スレッド数の優先度』が同じ種類であるマクロタスク」として扱う．

1. マクロタスク 1 とマクロタスク 2 のどちらも，*Relinearization* 及び *Rescaling* を含まない.

   - 常に 1 スレッドで実行される, つまり, 優先度は 1 スレッド使用についてのみ存在し, 優先度の値は 1 であるため.

2. 入力となる平文や暗号文のパラメータと，実行する準同型演算が同じである.

   - 各スレッド数での実行時間がおおよそ一致し，各スレッド数の優先度が同じとみなせるため.

「マクロタスクの各使用スレッド数の優先度」の種類数をMとする．実行待ちマクロタスク番号$i \neq i', \left(2 \leq i < i' \leq W(t)\right)$に対するマクロタスク$MT_{w_i^{(Job)}(t),w_i^{(MT)}(t)}$と$MT_{w_{i'}^{(Job)}(t),w_{i'}^{(MT)}(t)}$について，「マクロタスクの各使用スレッド数の優先度」の種類が同じであれば，式 2 における，$th^{(rel)}_{w_i^{(Job)}(t),w_i^{(MT)}(t)}\left(w_1^{(Job)}(t), w_1^{(MT)}(t), n\right)$と$th^{(rel)}_{w_{i'}^{(Job)}(t),w_{i'}^{(MT)}(t)}\left(w_1^{(Job)}(t), w_1^{(MT)}(t), n\right)$は同じである．「マクロタスクの各使用スレッド数の優先度」の種類ごとの実行待ちマクロタスク数を，各マクロタスクが（1）実行待ちマクロタスクリストに追加される時と，(2）実行開始される時に，それぞれ更新すれば,「マクロタスクの各使用スレッド数の優先度」の種類ごとの実行待ちマクロタスク数を用いることで, 式 2 における$\sum_{i=2}^{W(t)} th^{(rel)}_{w_i^{(Job)}(t),w_i^{(MT)}(t)}\left(w_1^{(Job)}(t), w_1^{(MT)}(t), n\right)$は，$M$回の乗算と$M-1$回の加算で求めることができる．そして，次に実行するマクロタスクに使用するスレッド数の候補数$\left|C_{w_1^{(Job)}(t),w_1^{(MT)}(t)}\right|$を用いて，最悪計算量は$O\left(M\left|C_{w_1^{(Job)}(t),w_1^{(MT)}(t)}\right|\right)$となる．$\left|C_{w_1^{(Job)}(t),w_1^{(MT)}(t)}\right|$が大きい場合は，二分探索を行うことで，$O\left(M\log_2\left|C_{w_1^{(Job)}(t),w_1^{(MT)}(t)}\right|\right)$とさらに小さくすることができる．また，実行待ちマクロタスクの数$W(t)$と使用可能なスレッド数$P^{(rest)}(t)$に対して，$W(t) \geq P^{(rest)}(t)$が真であれば，動的選択の処理を行わずに 1 スレッドのみ使用して次に実行するマクロタスクを実行開始するため，負荷が高い状況において，動的選択によるオーバヘッドをなくすことができる．まとめると，実行待ちマクロタスクが増加しても最悪計算量には影響を及ぼさずに，高速に使用スレッド数を選択することができる.

4.4 提案手法 DAMCREM の新規性

本節では，DAMCREM の新規性を示す．DAMCREM は，マクロタスクごとに予め使用スレッド数の候補を設定しておき，実行時にマクロタスクへの候補から選択することで，計算資源を動的に割り当てる．実行中のジョブ数や実行待ちマクロタスクの数に依存しないため，負荷が高い状況では，使用スレッド数の動的選択のオーバヘッドを小さくすることができる．負荷が低い状況でも，計算量を「マクロタスクの各使用スレッド数の優先度」の種類数と使用スレッド数の候補の積に抑えることができる．特に，完全準同型暗号を用いたアプリケーションでは，複数スレッドを割り当てる対象の準同型演算を *Relinearization* 及び *Rescaling* とした場合,「マクロタスクの各使用スレッド数の優先度」の種類数Mを，アプリケーションにおける最大レベルL_{Max}に対し

て，$M \leq 2L_{Max} + 1$とすることができる．使用スレッド数の動的選択のオーバヘッドを小さくすることで，ジョブ単位で計算資源を割り当てる処理を行うスレッドをジョブごとに用意する代わりに，より多くの CPU コアをマクロタスク実行に使用できる．

第5章　　評価実験

本章では，実施した評価実験と実験結果を示す．異なる平均クエリ到着間隔において，提案手法 DAMCREM を他の手法と比較することで，DAMCREM の利点及び改善すべき点を明らかにする．

まず，5.1 節で評価対象としたアプリケーションの概要について説明し，5.2 節で実験の条件を示す. そして, 5.4 節では多項式近似を, 5.5 節ではニューラルネットワークにおける推論処理を, それぞれ対象とするアプリケーションとして評価を行った.

5.1 評価対象としたアプリケーション

対象とするアプリケーションを変えて評価した理由は，マクロタスクグラフの形状によって，DAMCREM の効果を検証するためである. 2 つのアプリケーションを比較する. 多項式近似では, ニューラルネットワークにおける推論処理と比べて，マクロタスク間での演算内容の類似性が低く, マクロタスク間の並列性もニューラルネットワークにおける推論処理と比べると低い. 一方, ニューラルネットワークにおける推論処理では, 演算内容が似ているマクロタスクが複数存在し, マクロタスク間の並列性が高い．アプリケーションからマクロタスクグラフを作成する方法は 4.3.1 項で示した方法を用いた.

アプリケーションを多項式近似とニューラルネットワークにおける推論処理の 2 種類を選択した理由を下記に示す.

- 多項式近似：CKKS 方式では，暗号文を冪指数とした指数演算や対数演算，暗号文同士を除数とした除算を行うことができない．したがって，多項式近似で近似値を求めるのが一般的である[15], [17]ため.
- ニューラルネットワークにおける推論処理：完全準同型暗号を用いた機械学習に対する高速化を行う研究が盛んに行われている[7], [10], [11]ため.

5.2 実験条件

本節では，評価実験における条件について示す．なお，評価実験で使用した計算機は，表 2.2 で示した計算機である.

実験パラメータとして，総ジョブ数$B = 2048$とし，マクロタスクを実行するのに全体で使用可能なスレッド数$P = 63$とし，ジョブやマクロタスクの実行管理に 1 スレッドを使用した．他の計算機との通信は行わず，ジョブの実行に用いる入力暗号文は DRAM 上に予め確保するようにし

た．また，入力暗号文の生成から全てのジョブを実行し終えるまでを 1 セットとして，実験プログラムを 1 回実行した際に，連続して 11 セット繰り返した．試行間における誤差を小さくすることを目的に 11 セット繰り返したため，セット間でクエリ到着時刻s_iは同じとした．そして，最初の 1 セット目を除いた 2 セット目から 11 セット目を実験結果として用いた．また，各セットは，先頭 500 個のジョブと末尾 100 個のジョブを実験結果から除外した．セット開始直後とセット終了直前は，実行すべきジョブの数が少なくなるためである．

実験実施に必要な時間を削減するために，多項式近似アプリケーションとニューラルネットワークにおける推論処理アプリケーションのそれぞれに対して，十分に長いデッドラインを設けた．デッドラインを超えた場合，以降のジョブのレイテンシが短縮されることは無いと判断し，対象の実験パラメータと手法の組み合わせでの実験実施を中断し，実験結果から除外した．デッドラインは，各マクロタスクをシングルスレッドで逐次的に実行した時のレイテンシの 2 倍とした．2 倍とした理由は，乱数で生成したクエリ到着時刻s_iの間隔がクエリ到着平均クエリ到着間隔を下回る可能性があるためである．

表 5.1　評価実験で使用する記号の定義

記号	定義
P	● マクロタスク実行のために使用する総スレッド数($P = 63$)
B	● 評価対象であるジョブの総数($B = 2048$)
D	● ジョブのクエリ到着時刻の間隔[ms] ● 実験パラメータの 1 つである．
J_i	● i番目のジョブ ➢ $1 \leq i \leq B$ ➢ $1 \leq i < i'$の時，J_iのクエリ到着時刻は，$J_{i'}$のクエリ到着時刻よりも早い
s_i	● i番目のジョブに対応するクエリが到着する時刻[ms] ● $s_i = \begin{cases} 0 & (i = 1) \\ s_{i-1} + (-D\ln(1 - x_i)) & (1 \leq i \leq B) \end{cases}$ ただし，$x_i \in [0,1)$を満たす一様乱数で生成された値である．

5.3 比較対象の手法

提案手法の有用性を示すために，既存手法と比較を行う．既存手法として，下記の 2 種類の手法を用いた．

1. 各マクロタスクに使用するスレッド数を予め固定としたナイーブ手法
2. 実行待ちマクロタスクと利用可能なスレッド数に応じて，各マクロタスクに入力する暗号文のレベルに基づいて用意した使用スレッド数の候補から割り当てる手法（以下，レベル

ベース動的手法）[22]．ただし，DAMCREM よりも単純なルールで候補から選択する

5.3.1 ナイーブ手法

マクロタスクに使用するスレッド数を静的に決めておく手法をナイーブ手法とする．全てのマクロタスクに対して使用スレッド数を個別に設定すると，各マクロタスクに使用するスレッド数の組み合わせの総数が膨大となる．したがって，ナイーブ手法では，*Relinearization* か *Rescaling* を含む全てのマクロタスクに 1 スレッドのみを使用するといった典型的な組み合わせを使用する．具体的には，下記の 4 通りのいずれかを使用する．なお，使用可能なスレッド数が足りない場合は，使用スレッド数が使用可能になるまでマクロタスクの実行開始を遅らせる．1 つのマクロタスクに対して，ナイーブ手法 1 は最も割り当てるスレッド数が少なく，ナイーブ手法 2，ナイーブ手法 3 の順で割り当てるスレッド数が増え，ナイーブ手法 4 ではナイーブ手法の内，最も多くのスレッドを割り当てる．

ナイーブ手法1： 全てのマクロタスクに対して 1 スレッドのみ使用する．

ナイーブ手法2： *Relinearization* か *Rescaling* を含むマクロタスクには 2 スレッド使用し，それ以外のマクロタスクには 1 スレッドのみ使用する．

ナイーブ手法3： *Relinearization* を含むマクロタスクには$\lceil(l+2)/2\rceil$スレッド，*Rescaling* を含むマクロタスクには$\lceil(l+1)/2\rceil$スレッドをそれぞれ使用し，それ以外のマクロタスクには 1 スレッドのみ使用する．ただし，*Rescaling* を含むマクロタスクにおいて，入力暗号文がレベル$l=1$の場合は 2 スレッド使用する．

ナイーブ手法4： *Relinearization* を含むマクロタスクには$(l+2)$スレッド，*Rescaling* を含むマクロタスクには$(l+1)$スレッドをそれぞれ使用し，それ以外のマクロタスクには 1 スレッドのみ使用する．ただし，*Relinearization* を含むマクロタスクにおいて，入力暗号文のレベル$l=1$の場合は，2 スレッド使用する．

5.3.2 レベルベース動的手法

レベルベース動的手法は，提案手法 DAMCREM と同様に，マクロタスクごとに使用スレッド数の候補を用意しておき，実行時に候補から選択する手法である．DAMCREM とは，候補となるスレッド数を決定する方法と，実行時に候補から選択する基準が異なる．複雑な処理を行う DAMCREM と単純な処理を行うレベルベース動的手法を比べることで，DAMCREM の改善可能な点を探すために，レベルベース動的手法を用いる．具体的には，下記の 2 点が DAMCREM と異なる．

1. 静的な解析は行わず，使用スレッド数の候補をナイーブ手法と同様に，暗号文のレベルから決定する．

2. 実行待ちマクロタスクに残しておくスレッド数の見積もりにおいて，マクロタスクを下記の 2 種類に単純化した．

   I. *Relinearization* か *Rescaling* のどちらかを含むマクロタスク

   II. Relinearization と Rescaling のどちらも含まないマクロタスク

使用スレッド数の候補はマクロタスクを 3 種類に分けて用意する．

1. *Relinearization* を含むマクロタスク：$\{(l+2), \lceil (l+1)/2 \rceil, 2, 1\}$の 4 種類

2. *Rescaling* を含むマクロタスク：$\{(l+1), \lceil (l+1)/2 \rceil, 2, 1\}$の 4 種類

3. *Relinearization* と *Rescaling* のどちらも含まないマクロタスク：$\{1\}$の 1 種類

候補から使用するスレッド数の選択方法について説明する．*Relinearization* か *Rescaling* のどちらかを含むマクロタスクのそれぞれは，使用スレッド数の候補が 4 種類存在するため，表 5.2 に示す順番で条件判定をし，最初に条件を満たした使用スレッド数cを候補から選択する．ただし，実行待ちマクロタスクに使用可能なスレッド数を$P^{(rest)}(t)$とし，実行待ちマクロタスクリストの先頭を除くマクロタスクの内，*Relinearization* か *Rescaling* を含むマクロタスクの数を$n_r(t)$，*Relinearization* と *Rescaling* のどちらも含まないマクロタスクの数を$n_s(t)$とする．ナイーブ手法 3 及びナイーブ手法 4 と同様に，表 5.2 における条件 1 と条件 2 については，入力暗号文のレベル$l = 1$の場合は適用しない．

表 5.2　入力暗号文のレベルlに対するレベルベース動的手法の使用スレッド数選択方法

条件の番号（優先順位）	使用スレッド数の候補c		条件
	Relinearization	*Rescaling*	
1	$l+2$	$l+1$	$c + 2 \times n_r(t) + n_s(t) \leq P^{(rest)}(t)$ ※ただし，$l = 1$の場合を除く
2	$\left\lceil \frac{l+2}{2} \right\rceil$	$\left\lceil \frac{l+1}{2} \right\rceil$	$c + n_r(t) + n_s(t) \leq P^{(rest)}(t)$ ※ただし，$l = 1$の場合を除く
3	2	2	$c + n_r(t) + n_s(t) \leq P^{(rest)}(t)$
4	1	1	$c \leq P^{(rest)}(t)$

5.4 多項式近似を対象アプリケーションとした評価

5.4.1 使用したマクロタスクグラフ

7 次までの多項式近似のマクロタスクグラフを図 5.1 に示す．なお，マクロタスクの番号jが小さいほどジョブ内でのマクロタスクの優先度が高い．そして，ジョブのクエリとなる暗号文のレベルを8とした．

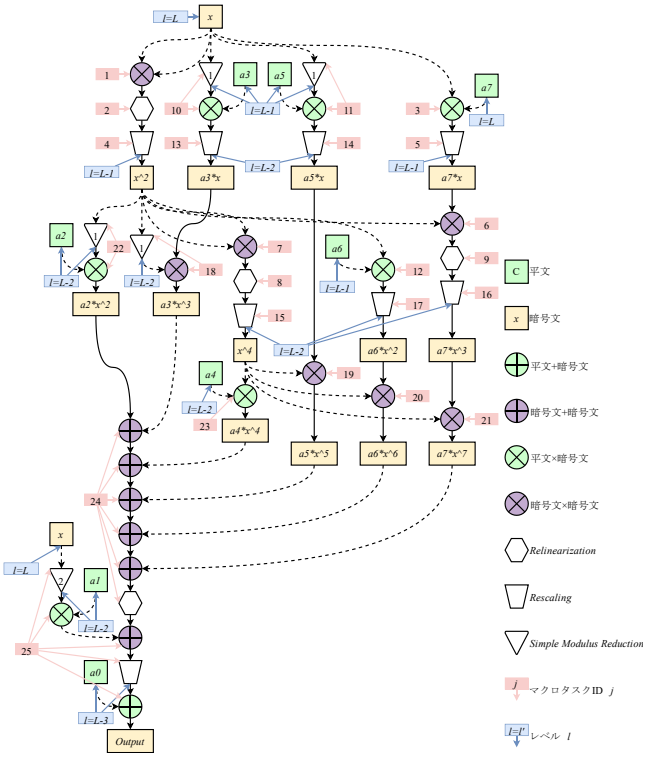

図 5.1　多項式近似のマクロタスクグラフ

DAMCREM における使用スレッド数の優先度の関係を示す．図 5.1 に示したマクロタスクグラフの各マクロタスクに対して，各スレッド数での実行時間を表 5.3 に示す．ただし，表 2.4 と表 2.5 で示した通り，使用スレッド数が *Relinearization* は$(l+2)$，Rescaling は$(l+1)$を超えると実行速度の向上率が低い．したがって，表 5.3 に示すスレッド数の上限はクエリの暗号文のレベルLを用いて，$(L+2)=10$スレッドとする．また，背景が灰色のセルは，使用するスレッド数に対して高速化の効果が小さいため，候補として使用しない．そして，表 5.3 で得られた値を元に，各マクロタスクに対する各使用スレッド数の優先度を計算し，マクロタスク間での使用スレッド数の関係を表 5.4 に示す．ただし，「あるマクロタスクに使用するスレッド数が 1 の場合，他の全てのマクロタスクにおいて，より高い優先度を持つスレッド数が存在しないため，他の全てのマクロタスクに残すスレッド数が 0 となる」というように，該当するスレッド数が自明である場合は表 5.4 に表記しない．さらに，マクロタスク番号jが$j \in \{4,5\}$のマクロタスクと，$j \in \{8,9\}$のマクロタスク，$j=\{13,14,15,16,17\}$のマクロタスクは，最大1.6[ms]の差は存在するものの，マクロタスクに含ま

れる準同型演算の種類や実行順序，入力暗号文のパラメータは同じであるため，表 5.4 では統合して示す. また, ナイーブ手法 1〜4 のスループットは, それぞれ16.5[ジョブ/秒], 14.2[ジョブ/秒], 12.2[ジョブ/秒]，10.3[ジョブ/秒]であった．スループットの算出方法について説明する．まず, 2048 個のジョブを同時に実行開始して, 各ジョブの実行完了時刻を求めることを 3 回繰り返した. そして，定常状態におけるスループットを求めるために，それぞれの試行において，スループットが安定していた，実行開始から 50 秒後から 100 秒後まで間における実行完了ジョブ数を求め, 平均することでスループットを算出した.

表 5.3　多項式近似アプリケーションを構成する
各マクロタスクに対する使用スレッド数と実行時間の関係

マクロタスク ID	使用スレッド数に対する実行時間[ms]									
j	1	2	3	4	5	6	7	8	9	10
1	42.3									
2	561.7	327.5	240.3	191.8	137.7	128.9	127.3	126.3	114.2	75.0
3	26.4									
4	180.5	94.3	68.3	51.6	46.9	36.7	36.2	30.7	26.6	
5	180.3	94.3	68.4	51.5	46.9	36.6	36.3	30.7	26.5	
6	52.4									
7	37.8									
8	464.5	281.6	185.4	162.6	126.6	117.3	115.9	104.1	69.4	
9	464.4	280.0	185.4	161.9	125.9	117.1	116.1	104.2	69.5	
10	23.4									
11	23.3									
12	23.2									
13	159.2	85.2	61.9	47.7	41.1	36.0	30.4	26.1		
14	159.3	83.2	61.8	47.0	40.5	36.0	30.4	26.1		
15	159.3	83.2	61.7	46.9	40.5	36.0	30.3	26.2		
16	159.5	83.5	61.8	46.8	40.5	36.0	30.4	26.2		
17	159.2	83.3	61.8	46.8	40.6	36.1	30.4	26.1		
18	47.0									
19	46.2									
20	46.1									
21	45.6									
22	20.5									
23	20.6									
24	394.2	237.3	177.7	135.0	125.6	123.3	110.9	82.0		
25	164.1	100.2	77.2	67.0	61.9	56.0	52.0			

表 5.4　多項式近似アプリケーションを構成する
各マクロタスク間の使用スレッド数の優先度の関係（その 1）

マクロタスク ID j'	使用スレッド数 n'	他のマクロタスク ID jのマクロタスク$MT_{i,j}$における対応するスレッド数$th_{i,j}^{(rel)}(i',j',n')$					
		2	4, 5	8, 9	13, 14, 15, 16, 17	24	25
2	2	1	2	1	2	1	1
	3	2	3	3	3	2	2
	4	3	4	3	4	3	3
	5	4	4	4	4	4	4
	6	5	6	5	6	5	5
	10	6	9	9	8	8	7
4, 5	2	1	1	1	1	1	1
	3	2	2	2	2	2	2
	4	3	3	3	3	3	3
	5	5	4	4	5	4	4
	6	5	5	5	5	5	4
	8	6	6	5	8	5	6
	9	6	8	5	8	8	7
8, 9	2	2	2	1	2	2	1
	3	2	3	2	3	2	2
	4	4	4	3	4	4	3
	5	5	5	4	5	4	4
	9	6	9	5	8	8	7

表 5.4　多項式近似アプリケーションを構成する
各マクロタスク間の使用スレッド数の優先度の関係（その 2）

マクロタスク ID j'	使用スレッド数 n'	他のマクロタスク ID jのマクロタスク$MT_{i,j}$における対応するスレッド数$th_{i,j}^{(rel)}(i',j',n')$					
		2	4, 5	8, 9	13, 14, 15, 16, 17	24	25
13, 14, 15, 16, 17	2	1	2	1	1	1	1
	3	2	3	2	2	2	2
	4	3	4	3	3	3	3
	5	5	4	4	4	4	4
	6	5	6	5	5	5	5
	7	6	6	5	6	5	5
	8	6	6	5	7	5	6
24	2	2	2	1	2	1	1
	3	3	3	3	3	2	2
	4	4	4	3	4	3	3
	5	5	5	5	5	4	4
	8	6	8	5	8	5	7
25	2	2	2	2	2	2	1
	3	3	3	3	3	3	2
	4	4	4	4	4	4	3
	5	5	6	5	5	5	4
	6	6	6	5	7	5	5
	7	6	8	5	8	5	6

5.4.2　実験結果

本項では, 多項式近似アプリケーションに対する実験結果を示す. 平均クエリ到着間隔60[ms], 70[ms]，80[ms]，90[ms]，100[ms]，110[ms]について，あるレイテンシ以下のジョブの割合を示した累積分布グラフをそれぞれ図 5.2～図 5.7 に示す．また，各手法におけるジョブのレイテンシの平均値と標準偏差，中央値，最大値，最小値を表 5.5～表 5.10 に示す．ただし，5.2 節で示した通り，平均クエリ到着間隔ごとに，ジョブのレイテンシの最大値がデッドラインを超えた手法は除外し，表 5.5～表 5.10 における各統計値を「—」と表記した．図 5.1 に示したマクロタスクグラフの各マクロタスクをシングルスレッドで逐次実行した際のレイテンシが3,661[ms]であったため，3,661[ms]の 2 倍である7,322[ms]をデッドラインとした.

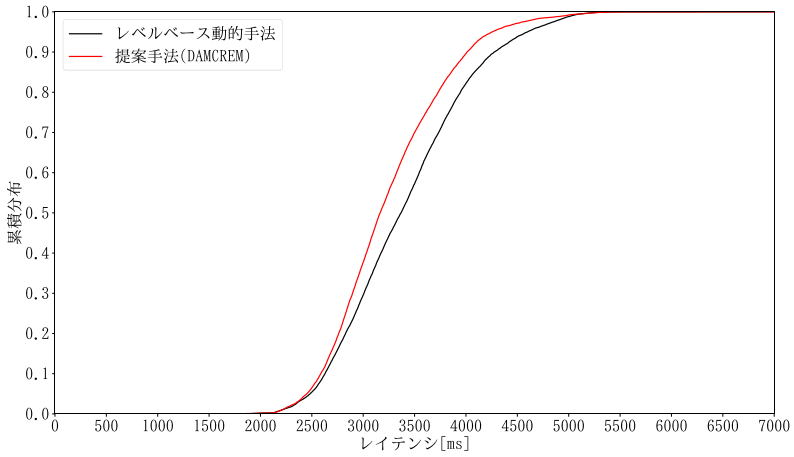

図 5.2　レイテンシが一定値以下のジョブ数の割合
（多項式近似，平均クエリ到着間隔60[ms]）

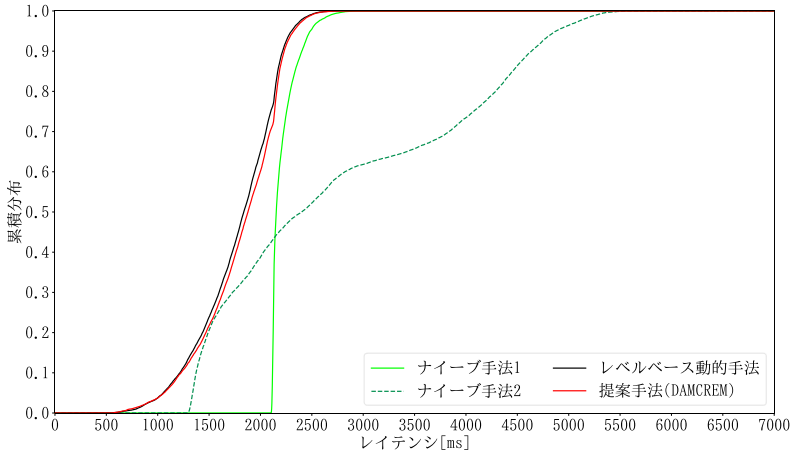

図 5.3　レイテンシが一定値以下のジョブ数の割合
（多項式近似，平均クエリ到着間隔70[ms]）

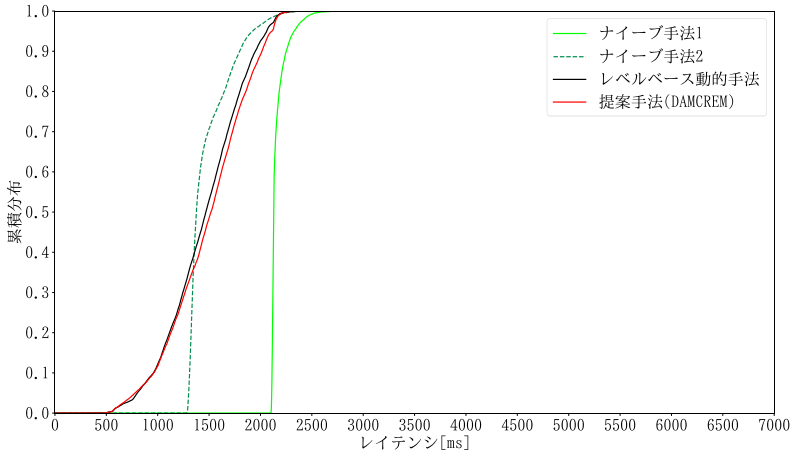

図 5.4　レイテンシが一定値以下のジョブ数の割合
（多項式近似，平均クエリ到着間隔80[ms]）

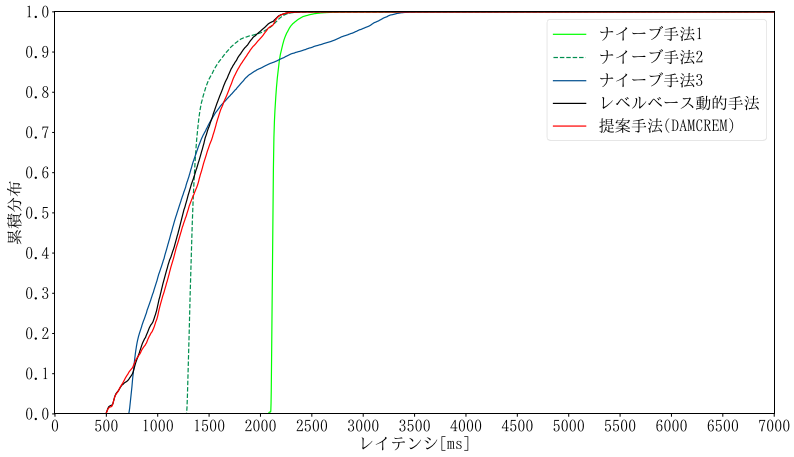

図 5.5　レイテンシが一定値以下のジョブ数の割合
（多項式近似，平均クエリ到着間隔90[ms]）

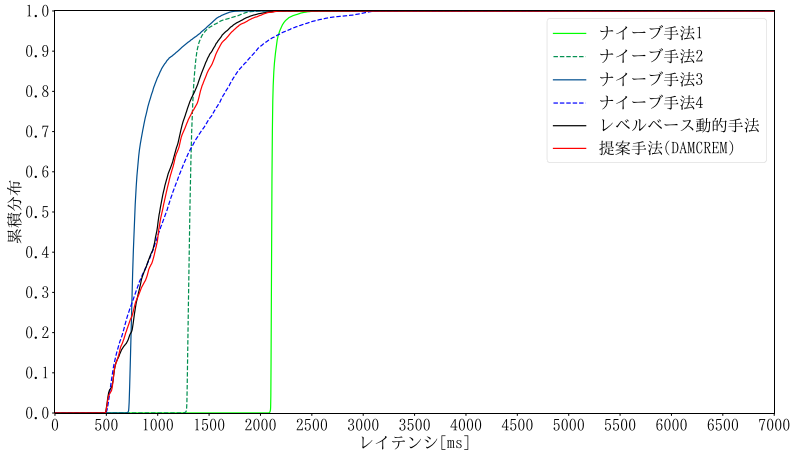

図 5.6　レイテンシが一定値以下のジョブ数の割合
（多項式近似，平均クエリ到着間隔100[ms]）

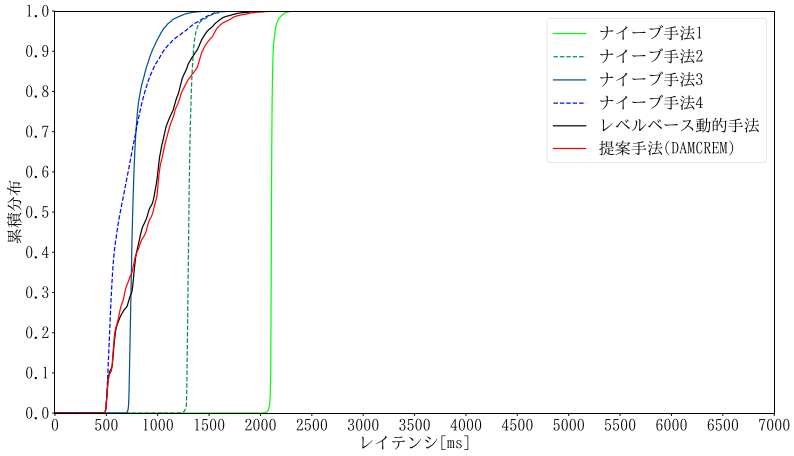

図 5.7　レイテンシが一定値以下のジョブ数の割合
（多項式近似，平均クエリ到着間隔110[ms]）

表 5.5　ジョブのレイテンシに対する統計値[ms]（多項式近似，平均クエリ到着間隔60[ms]）

手法	平均値	標準偏差	中央値	最大値	最小値
ナイーブ手法 1	—	—	—	—	—
ナイーブ手法 2	—	—	—	—	—
ナイーブ手法 3	—	—	—	—	—
ナイーブ手法 4	—	—	—	—	—
レベルベース動的手法	3,411	642	3,367	5,538	1,213
DAMCREM	3,248	572	3,165	5,610	1,670

※「—」は，ジョブのレイテンシがデッドラインである7,322[ms]を超えたため除外した．

表 5.6　ジョブのレイテンシに対する統計値[ms]（多項式近似，平均クエリ到着間隔70[ms]）

手法	平均値	標準偏差	中央値	最大値	最小値
ナイーブ手法 1	2,212	129	2,153	3,116	2,102
ナイーブ手法 2	2,788	1,271	2,400	5,616	1,291
ナイーブ手法 3	—	—	—	—	—
ナイーブ手法 4	—	—	—	—	—
レベルベース動的手法	1,780	393	1,838	2,901	510
DAMCREM	1,810	398	1,878	2,893	505

※「—」は，ジョブのレイテンシがデッドラインである7,322[ms]を超えたため除外した．

表 5.7　ジョブのレイテンシに対する統計値[ms]（多項式近似，平均クエリ到着間隔80[ms]）

手法	平均値値	標準偏差	中央値	最大値	最小値
ナイーブ手法 1	2,160	77	2,129	2,894	2,099
ナイーブ手法 2	1,474	210	1,376	2,453	1,282
ナイーブ手法 3	—	—	—	—	—
ナイーブ手法 4	—	—	—	—	—
レベルベース動的手法	1,455	378	1,469	2,569	493
DAMCREM	1,489	395	1,518	2,545	488

※「—」は，ジョブのレイテンシがデッドラインである7,322[ms]を超えたため除外した．

表 5.8　ジョブのレイテンシに対する統計値[ms]（多項式近似，平均クエリ到着間隔90[ms]）

手法	平均値	標準偏差	中央値	最大値	最小値
ナイーブ手法 1	2,142	65	2,121	2,936	2,073
ナイーブ手法 2	1,421	209	1,340	2,417	1,278
ナイーブ手法 3	1,364	635	1,188	3,537	710
ナイーブ手法 4	—	—	—	—	—
レベルベース動的手法	1,267	405	1,247	2,585	489
DAMCREM	1,304	424	1,287	2,404	489

※「—」は，ジョブのレイテンシがデッドラインである7,322[ms]を超えたため除外した．

表 5.9　ジョブのレイテンシに対する統計値[ms]（多項式近似，平均クエリ到着間隔100[ms]）

手法	平均値	標準偏差	中央値	最大値	最小値
ナイーブ手法 1	2,122	42	2,110	2,570	2,080
ナイーブ手法 2	1,336	84	1313	2,032	1,239
ナイーブ手法 3	871	215	778	1,817	701
ナイーブ手法 4	1,187	555	1,084	3,221	484
レベルベース動的手法	1,047	351	1,019	2,393	476
DAMCREM	1,071	374	1,043	2,364	480

表 5.10　ジョブのレイテンシに対する統計値[ms]（多項式近似，平均クエリ到着間隔110[ms]）

手法	平均値	標準偏差	中央値	最大	最小値
ナイーブ手法 1	2,111	21	2,107	2,428	2,017
ナイーブ手法 2	1,314	42	1,303	1,743	1,217
ナイーブ手法 3	798	109	756	1,486	683
ナイーブ手法 4	722	238	635	1,678	474
レベルベース動的手法	929	312	909	2,170	478
DAMCREM	948	342	954	2,228	476

平均クエリ到着間隔が長くなるにつれて生じた，結果の変化について説明する．

平均クエリ到着間隔$D = 60$[ms]

DAMCREM とレベルベース動的手法のみデッドラインを守ることができた．ナイーブ手法 2〜4 は，負荷が高いにも関わらず，マクロタスクに複数スレッドを使用したために，スループットが低下し，デッドラインを超えた．また，DAMCREM とレベルベース動的手法では，負荷が高い場合には使用するスレッド数を減らした一方で，実験開始直後やクエリ到着時刻s_iの間隔が長く，実行すべきマクロタスク数が少ない時には，ナイーブ手法 1 と異なり，マクロタスクに複数スレッドを使用することで，未使用のスレッド数を削減したため，デッドラインを守ることができたと考えられる．

DAMCREM は，レベルベース動的手法と比べると平均値や標準偏差，中央値が小さいが，最大値や最小値が大きい．これは，DAMCREM は，レベルベース動的手法よりも多くのスレッドを割り当てにくいためと考えられる．

平均クエリ到着間隔$D = 70$[ms]

DAMCREM とレベルベース動的手法，ナイーブ手法 1，ナイーブ手法 2 がデッドラインを守ることができた．DAMCREM とレベルベース動的手法は，レイテンシの標準偏差が大きいが，平均値や中央値，最大値，最小値のいずれもナイーブ手法 1 よりも小さい．また，DAMCREM とレベルベース動的手法はナイーブ手法 2 と比べて，いずれの統計値も小さい．

DAMCREM は，レベルベース動的手法を比べると，平均クエリ到着間隔が60[ms]の結果と比べて，平均値や標準偏差，中央値が大きいが，最大値や最小値が小さい．平均クエリ到着間隔が60[ms]よりも負荷が低くなったために，多くのスレッドを割り当てやすいレベル

ベース動的手法が平均値や中央値を小さくなったと考えられる．

平均クエリ到着間隔$D = 80$[ms]

DAMCREM とレベルベース動的手法，ナイーブ手法 1，ナイーブ手法 2 がデッドラインを守ることができた．DAMCREM とレベルベース動的手法は，ナイーブ手法 2 と比べて，レイテンシの最小値が小さい一方で，平均値，標準偏差，最大値が大きい．これは，DAMCREM とレベルベース動的手法では，実行待ちマクロタスクの数が少ない場合には，ナイーブ手法 2 と比べて多くのスレッドを割り当てることがある．しかし，DAMCREM とレベルベース動的手法では，新たに追加されるマクロタスクを考慮していない．多くのスレッドを割り当てたことで，利用可能スレッド数が減少し，多くのスレッドを割り当てたマクロタスクが実行完了した際に利用可能スレッド数が大きく増加したために，使用スレッド数が安定しなかったことが理由として考えられる．

平均クエリ到着間隔$D = 90$[ms]

DAMCREM とレベルベース動的手法，ナイーブ手法 1～3 がデッドラインを守ることができた．DAMCREM とレベルベース動的手法は，ナイーブ手法 2 やナイーブ手法 3 と比べて，レイテンシの平均値は小さいが，中央値は大きい．これは，DAMCREM とレベルベース動的手法と比べ，ナイーブ手法 2 は最小値が大きく，多くのジョブが中央値と同等のレイテンシであったことと，ナイーブ手法 3 はレイテンシが高いジョブが多かったためである．

平均クエリ到着間隔$D = 100$[ms]

全ての手法がデッドラインを守ることができた．DAMCREM とレベルベース動的手法は，ナイーブ手法 4 と比べて，いずれの統計値も小さい．これは，常にマクロタスクに最大スレッド数を割り当てるには，負荷が十分に低くないためである．また，全手法を比較すると，ナイーブ手法 3 が最も平均値や中央値が小さい．

平均クエリ到着間隔$D = 110$[ms]

全ての手法がデッドラインを守ることができた．ナイーブ手法 4 は，DAMCREM とレベルベース動的手法と比べて，いずれの統計値も小さい．これは，平均クエリ到着間隔が十分に長く，負荷が低い状態であることを示している．DAMCREM とレベルベース動的手法のレイテンシの平均値や中央値がナイーブ手法 3 やナイーブ手法 4 と比べて大きい理由として，実行待ちマクロタスクに対して残したスレッド数が過剰である状態が生じていたためと考えられる．

5.4.3 実験結果のまとめ

平均クエリ到着間隔間でのレイテンシの平均値の比較を図 5.8 と表 5.11 に示す．平均クエリ到着間隔Dが60[ms]及び70[ms]では，DAMCREM とレベルベース動的手法は，ナイーブ手法よりもレイテンシの平均値や中央値が小さい．ナイーブ手法 1 やナイーブ手法 2 と違い，負荷が一時的に低くなった際に，1 つのマクロタスクに対して多くのスレッドを割り当てることができるためと考えられる．一方，平均クエリ到着間隔Dが100[ms]以上では， DAMCREM とレベルベース動的手法よりもナイーブ手法 3 よりもレイテンシの平均値が 20.2～31.3%長い．平均クエリ到着間隔

Dが長くなるにつれて，DAMCREM とレベルベース動的手法の性能は 1 つのマクロタスクに多数のスレッドを使用するナイーブ手法と比べて劣っている．理由として，実行待ちマクロタスクのために残しておくスレッド数の見積もりの精度が不十分であり，使用スレッド数が安定していないことが考えられる．

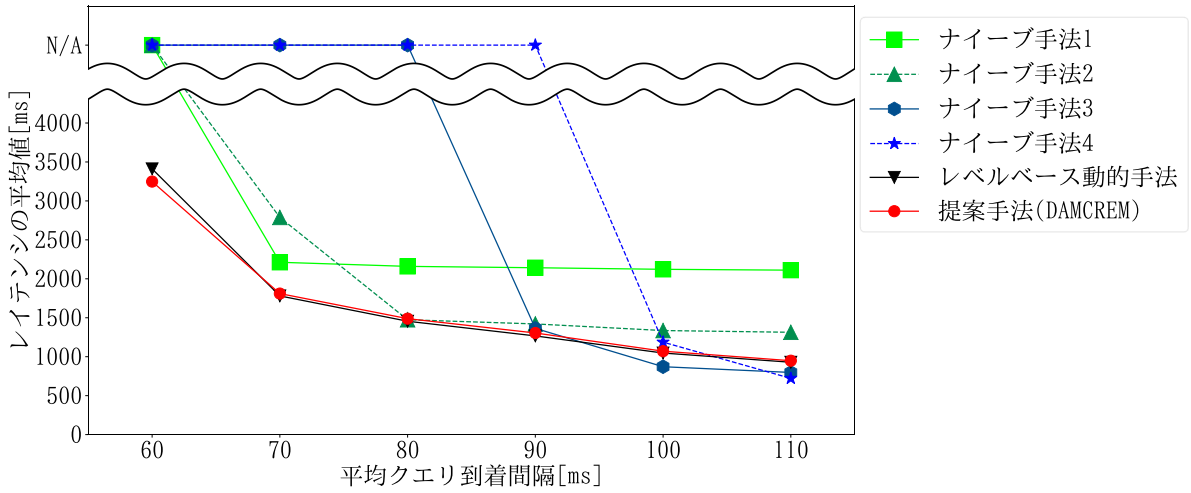

図 5.8　平均クエリ到着間隔とレイテンシの平均値の関係（多項式近似）

※「N/A」は，ジョブのレイテンシがデッドラインである7,322[ms]を超えたことを示す．

表 5.11　平均クエリ到着間隔とレイテンシの平均値の関係（多項式近似）

手法	平均クエリ到着間隔D[ms]に対するレイテンシの平均値					
	$D = 60$	$D = 70$	$D = 80$	$D = 90$	$D = 100$	$D = 110$
ナイーブ手法 1	—	2,212	2,160	2,142	2,122	2,111
ナイーブ手法 2	—	2,788	1,474	1,421	1,336	1,314
ナイーブ手法 3	—	—	—	1,364	871	798
ナイーブ手法 4	—	—	—	—	1,187	722
レベルベース動的手法	3,411	1,780	1,455	1,267	1,047	929
DAMCREM	3,248	1,810	1,489	1,304	1,071	948

※「—」は，ジョブのレイテンシがデッドラインである7,322[ms]を超えたため除外した．

5.5 ニューラルネットワークにおける推論処理を対象アプリケーションとした評価

5.5.1　使用したマクロタスクグラフ

使用したマクロタスクグラフを図 5.9 に示す．

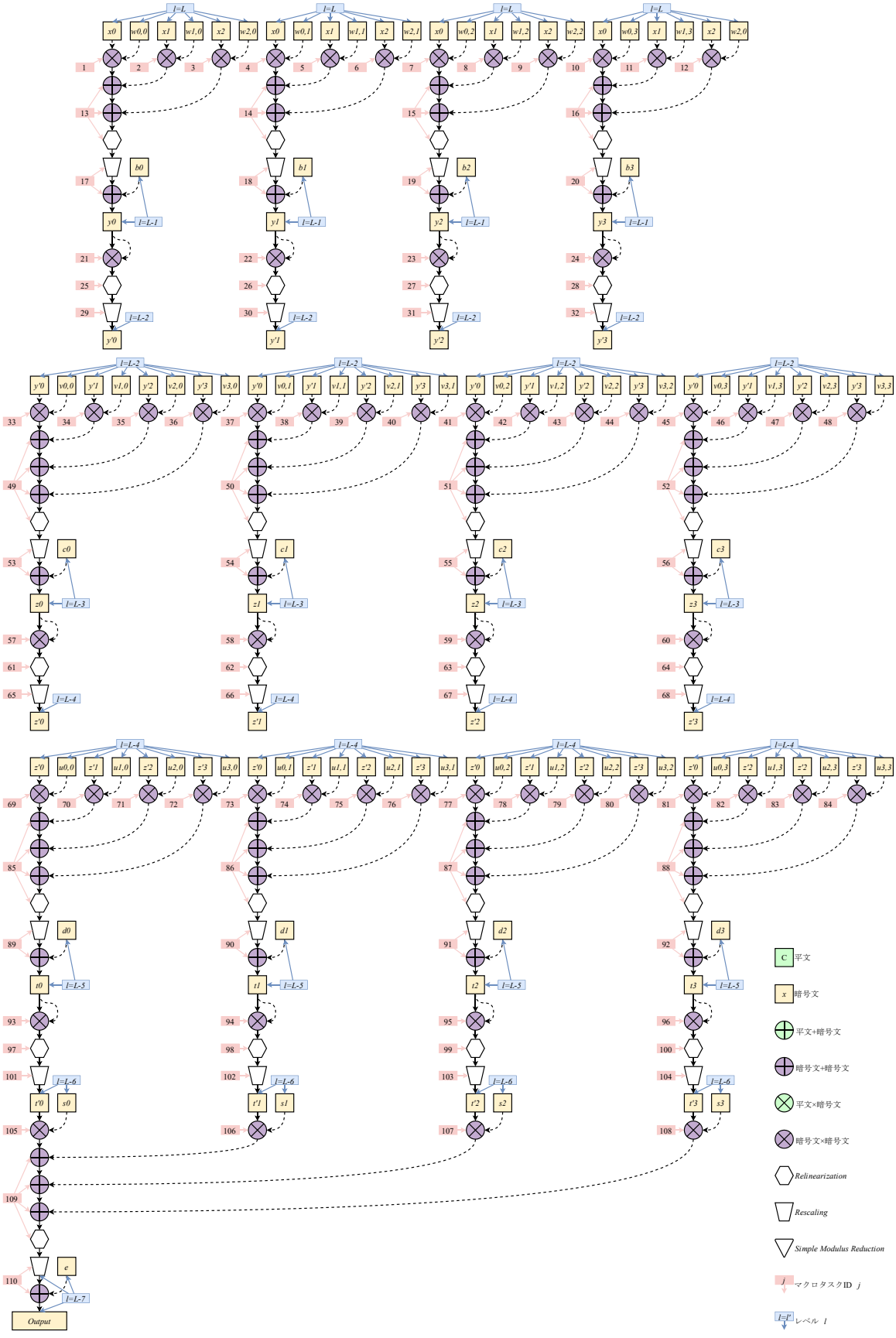

図 5.9　ニューラルネットワークにおける推論処理のマクロタスクグラフ

各マクロタスクの実行時間を表 5.12 に示す．ただし，マクロタスクの数が多いため，入力暗号文のパラメータが同じかつ実行する準同型演算の種類が同じマクロタスクごとにまとめて平均値を示す．具体的には，任意のジョブ IDiに対して，$\{MT_{i,1},\cdots,MT_{i,12}\}$，$\{MT_{i,13},\cdots,MT_{i,16}\}$，$\{MT_{i,17},\cdots,MT_{i,20}\}$，$\{MT_{i,21},\cdots,MT_{i,24}\}$，$\{MT_{i,25},\cdots,MT_{i,28}\}$，$\{MT_{i,29},\cdots,MT_{i,32}\}$，$\{MT_{i,33},\cdots,MT_{i,48}\}$，$\{MT_{i,49},\cdots,MT_{i,52}\}$，$\{MT_{i,53},\cdots,MT_{i,56}\}$，$\{MT_{i,57},\cdots,MT_{i,60}\}$，$\{MT_{i,61},\cdots,MT_{i,64}\}$，$\{MT_{i,65},\cdots,MT_{i,68}\}$，$\{MT_{i,69},\cdots,MT_{i,84}\}$，$\{MT_{i,85},\cdots,MT_{i,88}\}$，$\{MT_{i,89},\cdots,MT_{i,92}\}$，$\{MT_{i,93},\cdots,MT_{i,96}\}$，$\{MT_{i,97},\cdots,MT_{i,100}\}$，$\{MT_{i,101},\cdots,MT_{i,104}\}$，$\{MT_{i,105},\cdots,MT_{i,108}\}$，$\{MT_{i,109}\}$，$\{MT_{i,110}\}$をそれぞれ示す．なお，5.4.1 項と同様に，表 5.12 において，初期レベル$L=7$としてスレッド数の上限を 9 とし，背景が灰色のセルは，使用するスレッド数に対して高速化の効果が小さいため，候補として使用しない．表 5.12 の値から各マクロタスク間における使用スレッド数の各候補の優先度の関係を表 5.13 に示す．ただし，常に 1 スレッドとなる組み合わせは省略した．ナイーブ手法 1〜4 について，5.4.1 項での多項式近似と同様にして算出したスループットは，それぞれ5.2[ジョブ/秒]，4.5[ジョブ/秒]，4.1[ジョブ/秒]，3.5[ジョブ/秒]であった．

表 5.12　ニューラルネットワークにおける推論処理アプリケーションを構成する各マクロタスクに対する使用スレッド数と実行時間の関係

マクロタスク ID	使用スレッド数に対する実行時間[ms]								
j	1	2	3	4	5	6	7	8	9
1～12	53.5								
13～16	473.6	289.0	195.1	170.7	135.6	126.7	125.2	112.7	79.4
17～20	161.6	85.7	64.2	49.4	43.2	38.7	32.8	28.6	
21～24	31.7								
25～28	376.3	216.0	160.0	116.3	107.3	106.3	93.4	64.2	
29～32	137.9	72.3	51.0	40.4	35.8	29.9	25.8		
33～48	40.2								
49～52	306.7	190.3	141.0	107.0	105.8	93.0	68.7		
53～56	118.2	63.2	48.1	37.5	31.5	27.2			
57～60	22.6								
61～64	226.2	131.1	96.6	84.1	70.9	53.6			
65～68	95.3	49.7	39.1	29.5	24.9				
69～84	26.9								
85～88	171.5	107.2	82.5	69.2	55.4				
89～92	75.0	40.6	30.2	25.8					
93～96	13.7								
97～100	113.0	67.4	52.1	43.2					
101～104	52.8	28.6	24.3						
105～108	13.2								
109	73.7	44.7							
110	31.7	18.0							

表 5.13　ニューラルネットワークにおける推論処理アプリケーションを構成する各マクロタスク間の使用スレッド数の優先度の関係（その 1）

マクロタスク ID j'	使用スレッド数n'	他方のマクロタスク ID jの マクロタスク$MT_{i,j}$におけるスレッド数$th_{i,j}^{(rel)}(i',j',n')$													
		13, 14, 15, 16	17, 18, 19, 20	25, 26, 27, 28	29, 30, 31, 32	49, 50, 51, 52	53, 54, 55, 56	61, 62, 63, 64	65, 66, 67, 68	85, 86, 87, 88	89 90, 91, 92	97, 98, 99, 100	101, 102, 103, 104	109	110
13,14,15,16	2	1	2	2	2	1	2	2	2	1	2	2	2	2	2
	3	2	3	2	3	2	3	2	3	2	3	2	2	2	2
	4	3	4	4	4	4	4	3	4	3	4	3	4	3	3
	5	4	5	4	5	4	5	4	5	4	4	4	4	4	4
	9	5	8	8	7	7	6	6	5	5	4	4	4	4	4
17,18,19,20	2	1	1	1	2	1	1	1	2	1	1	1	1	1	1
	3	2	2	2	3	2	2	2	2	2	2	2	2	2	2
	4	3	3	3	4	3	3	3	3	3	3	3	3	2	2
	5	4	4	4	5	4	5	4	5	4	4	4	4	3	4
	6	5	5	4	6	4	6	6	5	5	4	4	4	4	4
	7	5	6	4	7	4	6	6	5	5	4	4	4	4	4
	8	5	7	8	7	7	6	6	5	5	4	4	4	4	4
25,26,27,28	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2
	3	3	3	2	3	2	3	2	3	2	3	2	2	2	2
	4	3	4	3	4	3	3	3	3	3	3	3	3	2	2
	8	5	7	4	7	7	6	6	5	5	4	4	4	4	4
29,30,31,32	2	1	1	1	1	1	1	1	2	1	1	1	1	1	1
	3	2	2	2	2	2	2	2	2	2	2	2	2	2	2
	4	3	3	3	3	3	3	3	3	3	3	3	3	2	2
	5	4	4	4	4	4	4	4	4	4	4	4	4	3	4
	6	5	5	4	5	4	5	5	5	5	4	4	4	4	4
	7	5	6	4	6	4	6	6	5	5	4	4	4	4	4
49,50,51,52	2	2	2	2	2	1	2	2	2	1	2	2	2	2	2
	3	3	3	3	3	2	3	3	3	2	3	2	3	2	2
	4	3	4	4	4	3	4	3	4	3	4	3	4	3	2
	7	5	7	4	7	4	6	6	5	5	4	4	4	4	4
53,54,55,56	2	1	2	1	2	1	1	1	2	1	1	1	1	1	1
	3	2	3	2	3	2	2	2	2	2	3	2	2	2	2
	4	3	4	4	4	3	3	3	4	3	3	3	3	2	2
	5	4	4	4	5	4	4	4	5	4	4	4	4	3	4
	6	5	5	4	6	4	5	5	5	5	4	4	4	4	4

表 5.13　ニューラルネットワークにおける推論処理アプリケーションを構成する各マクロタスク間の使用スレッド数の優先度の関係（その 2）

マクロタスク ID j'	使用スレッド数n'	他方のマクロタスク ID jの マクロタスク$MT_{i,j}$におけるスレッド数$th_{i,j}^{(rel)}(i',j',n')$													
		13, 14, 15, 16	17, 18, 19, 20	25, 26, 27, 28	29, 30, 31, 32	49, 50, 51, 52	53, 54, 55, 56	61, 62, 63, 64	65, 66, 67, 68	85, 86, 87, 88	89 90, 91, 92	97, 98, 99, 100	101, 102, 103, 104	109	110
61,62,63,64	2	1	2	2	2	1	2	1	2	1	2	1	2	1	2
	3	3	3	3	3	2	3	2	3	2	3	2	2	2	2
	4	4	4	4	4	4	4	3	4	3	4	3	4	3	3
	5	5	5	4	5	4	5	4	5	4	4	4	4	4	4
	6	5	5	4	6	4	6	5	5	5	4	4	4	4	4
65,66,67,68	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	3	2	3	2	3	2	3	2	2	2	3	2	2	2	2
	4	3	4	4	4	3	3	3	3	3	3	3	3	2	3
	5	4	4	4	5	4	4	4	4	4	4	4	4	3	4
85,86,87,88	2	2	2	2	2	2	2	2	2	1	2	2	2	2	2
	3	3	3	3	3	3	3	3	3	2	3	3	3	2	2
	4	4	4	4	4	4	4	4	4	3	4	4	4	3	3
	5	5	5	4	5	4	5	5	5	4	4	4	4	4	4
89,90,91,92	2	1	2	1	2	1	2	1	2	1	1	1	2	1	1
	3	2	3	2	3	2	2	2	2	2	2	2	2	2	2
	4	3	4	4	4	3	4	3	4	3	3	3	3	2	2
97,98,99,100	2	1	2	2	2	1	2	2	2	1	2	1	2	1	2
	3	3	3	3	3	3	3	3	3	2	3	2	3	2	2
	4	4	4	4	4	4	4	4	4	3	4	3	4	3	3
101,102, 103,104	2	1	2	1	2	1	2	1	2	1	1	1	1	1	1
	3	3	3	3	3	2	3	3	3	2	3	2	2	2	2
	4	3	4	4	4	3	4	3	4	3	4	3	3	2	2
109	2	1	2	2	2	1	2	2	2	1	2	2	2	1	2
110	2	1	2	1	2	1	2	1	2	1	2	1	2	1	1

5.5.2　実験結果

本項では，ニューラルネットワークにおける推論処理アプリケーションに対する実験結果を示す．平均クエリ到着間隔200[ms]，240[ms]，280[ms]，320[ms]，360[ms]について，あるレイテンシ以下のジョブの割合を示した累積分布グラフをそれぞれ図 5.10～図 5.13 に示す．また，各手法におけるジョブのレイテンシの平均値と標準偏差，中央値，最大値，最小値を表 5.14～表 5.17

に示す．ただし，5.2 節で示した通り，平均クエリ到着間隔ごとに，ジョブのレイテンシの最大値がデッドラインを超えた手法は除外し，表 5.14～表 5.17 における各統計値を「—」と表記した．図 5.9 に示したマクロタスク食いラフの各マクロタスクをシングルスレッドで逐次実行した際のレイテンシが11,379[ms]であったため，11,379[ms]の 2 倍である22,758[ms]をデッドラインとした．ただし，デッドラインを守ることができた手法では，レイテンシの最大値が9,000[ms]以下であったため，図 5.10～図 5.13 におけるレイテンシの最大値は9,000[ms]とした．

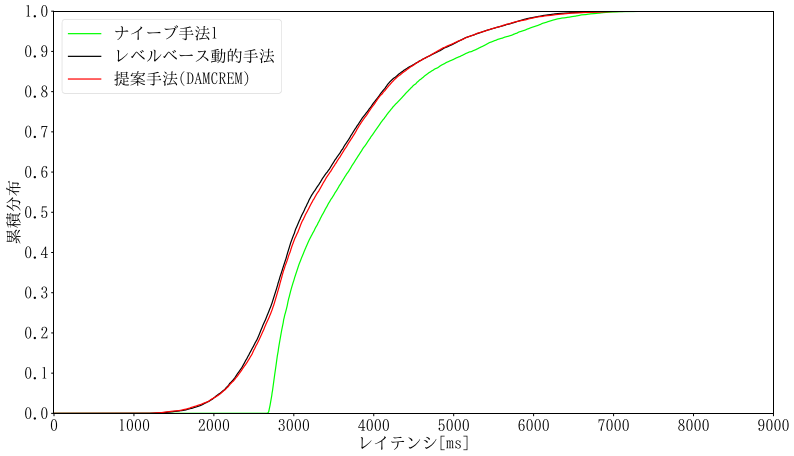

図 5.10　レイテンシが一定値以下のジョブ数の割合
（ニューラルネットワークにおける推論処理，平均クエリ到着間隔200[ms]）

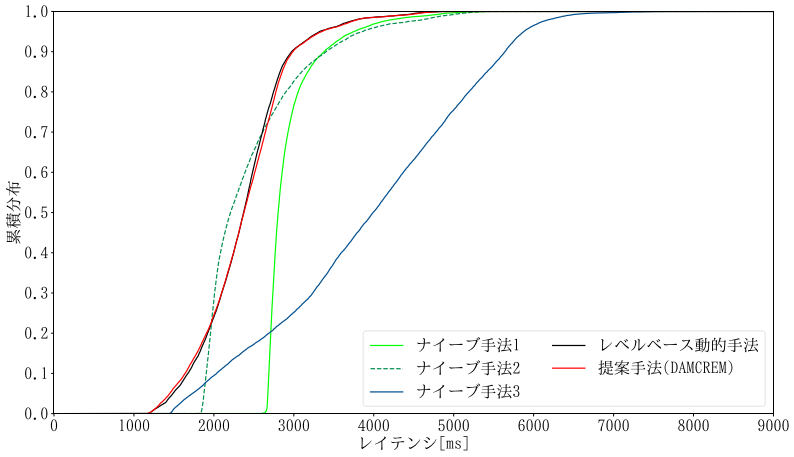

図 5.11　レイテンシが一定値以下のジョブ数の割合
（ニューラルネットワークにおける推論処理，平均クエリ到着間隔240[ms]）

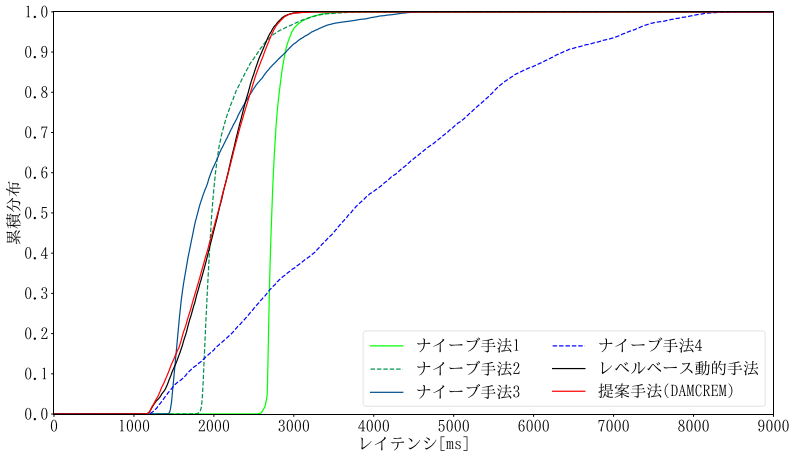

図 5.12　レイテンシが一定値以下のジョブ数の割合
（ニューラルネットワークにおける推論処理，平均クエリ到着間隔280[ms]）

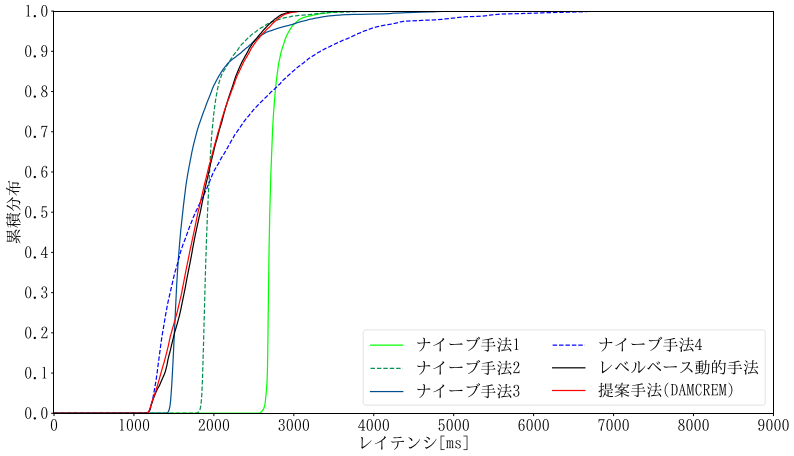

図 5.13　レイテンシが一定値以下のジョブ数の割合
（ニューラルネットワークにおける推論処理，平均クエリ到着間隔320[ms]）

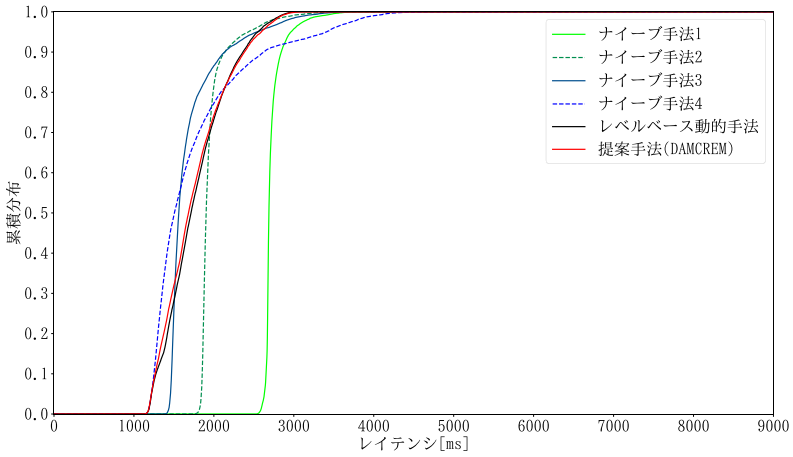

図 5.14　レイテンシが一定値以下のジョブ数の割合
（ニューラルネットワークにおける推論処理，平均クエリ到着間隔360[ms]）

表 5.14　ジョブのレイテンシに対する統計値[ms]
（ニューラルネットワークにおける推論処理，平均クエリ到着間隔200[ms]）

手法	平均値	標準偏差	中央値	最大値	最小値
ナイーブ手法 1	3,692	982	3,375	7,594	2,653
ナイーブ手法 2	—	—	—	—	—
ナイーブ手法 3	—	—	—	—	—
ナイーブ手法 4	—	—	—	—	—
レベルベース動的手法	3,362	988	3,120	7,236	1,179
DAMCREM	3,386	989	3,163	7,523	1,180

※「—」は，ジョブのレイテンシがデッドラインである22,758[ms]を超えたため除外した．

表 5.15　ジョブのレイテンシに対する統計値[ms]
（ニューラルネットワークにおける推論処理，平均クエリ到着間隔240[ms]）

手法	平均値	標準偏差	中央値	最大値	最小値
ナイーブ手法 1	2,942	382	2,804	5,528	2,579
ナイーブ手法 2	2,467	670	2,208	5,516	1,790
ナイーブ手法 3	3,942	1,298	3,991	7,817	1,434
ナイーブ手法 4	—	—	—	—	—
レベルベース動的手法	2,376	560	2,373	5,016	1,152
DAMCREM	2,381	578	2,376	5,101	1,149

※「—」は，ジョブのレイテンシがデッドラインである22,758[ms]を超えたため除外した．

表 5.16　ジョブのレイテンシに対する統計値[ms]
（ニューラルネットワークにおける推論処理，平均クエリ到着間隔280[ms]）

手法	平均値	標準偏差	中央値	最大値	最小値
ナイーブ手法 1	2,761	119	2,724	3,898	2,551
ナイーブ手法 2	2,104	316	1,975	3,965	1,769
ナイーブ手法 3	2,029	597	1,798	4,648	1,381
ナイーブ手法 4	3,911	1,754	3,712	8,617	1,128
レベルベース動的手法	2,044	418	2,055	3,308	1,135
DAMCREM	2,039	440	2,050	3,382	1,132

表 5.17　ジョブのレイテンシに対する統計値[ms]
（ニューラルネットワークにおける推論処理，平均クエリ到着間隔320[ms]）

手法	平均値	標準偏差	中央値	最大値	最小値
ナイーブ手法 1	2,736	111	2,700	3,766	2,559
ナイーブ手法 2	2,006	248	1,921	4,062	1,777
ナイーブ手法 3	1,787	456	1,610	4,998	1,404
ナイーブ手法 4	2,085	916	1,771	6,787	1,148
レベルベース動的手法	1,870	395	1,830	3,327	1,146
DAMCREM	1,856	414	1,802	3,316	1,145

表 5.18　ジョブのレイテンシに対する統計値[ms]
（ニューラルネットワークにおける推論処理，平均クエリ到着間隔360[ms]）

手法	平均値	標準偏差	中央値	最大値	最小値
ナイーブ手法 1	2,728	128	2,688	3,834	2,530
ナイーブ手法 2	1,970	210	1,903	3,411	1,756
ナイーブ手法 3	1,701	355	1,561	3,733	1,382
ナイーブ手法 4	1,758	649	1,506	4,687	1,130
レベルベース動的手法	1,777	401	1,714	3,100	1,120
DAMCREM	1,760	417	1,682	3,107	1,125

平均クエリ到着間隔$D = 200$[ms]

DAMCREM とレベルベース動的手法，ナイーブ手法 1 がデッドラインを守ることができた．DAMCREM とレベルベース動的手法は，ナイーブ手法 1 よりもレイテンシを短縮することができた．レイテンシに差が生じた理由として，1 つのジョブあたりのマクロタスクの並列性が高いことにより，負荷が変動しやすいことが考えられる．具体的に説明する．クエリ到着間隔が短くなると，実行待ちマクロタスク数が大きく増加し，逆にクエリ到着間隔が長くなると，実行待ちマクロタスク数が大きく減少する. クエリ到着間隔が短くなった場合は負荷が高いため, DAMCREM 及びレベルベース動的手法，ナイーブ手法 1 はいずれも，1 つのマクロタスクに 1 スレッドのみ

割り当てて実行する．しかし，クエリ到着間隔が長くなった場合には，ナイーブ手法 1 は，1 つのマクロタスクに 1 スレッドのみ割り当てるのに対して，DAMCREM とレベルベース動的手法では，1 つのマクロタスクに複数のスレッドを割り当てる．したがって，図 5.15 に示したジョブ ID が 600 付近や 800 付近のように，DAMCREM とレベルベース動的手法では，負荷が低い状況でレイテンシを短縮することができた．また，レベルベース動的手法におけるレイテンシの最大値が DAMCREM におけるレイテンシの最大値より低い理由として，レベルベース動的手法は DAMCREM よりも多くのスレッドを割り当てやすいことが考えられる．具体的には，負荷が低い状況で多くのスレッドを用いてマクロタスクを実行でき，実行待ちマクロタスクの数を少なく保つことで，負荷が一時的に高くなっても，レイテンシの増加を抑えることができる．また，実行待ちマクロタスクの数と利用可能なスレッド数が多い状況では，DAMCREM は，多くのスレッドを実行待ちマクロタスクのために残すため，各実行待ちマクロタスクが実行されるまで利用されないスレッド数がレベルベース動的手法よりも多くなる．

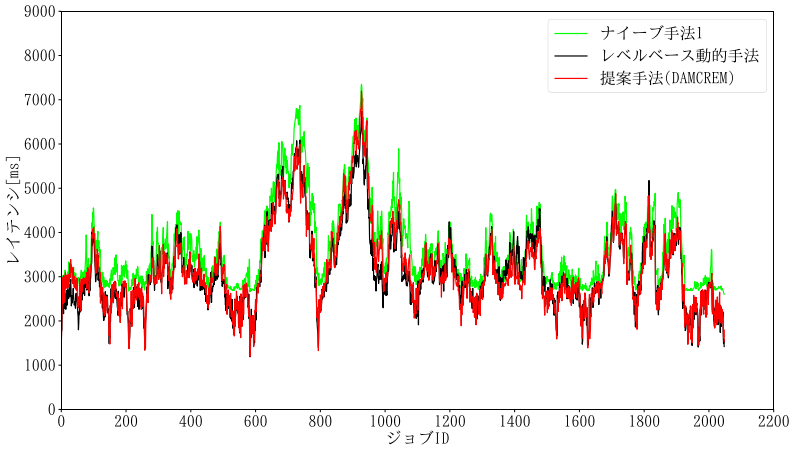

図 5.15　ジョブごとのレイテンシ（ニューラルネットワークにおける推論処理，平均クエリ到着間隔$D = 200$[ms]，11 セット目）

平均クエリ到着間隔$D = 240$[ms]

DAMCREM とレベルベース動的手法, ナイーブ手法 1～3 がデッドラインを守ることができた．ナイーブ手法 3 は，1 つのマクロタスクに多数のスレッドを割り当てるため，レイテンシが安定しなかった．ナイーブ手法 2 は DAMCREM やレベルベース動的手法と比べて，平均値や分散は大きいが，中央値は小さい．平均クエリ到着間隔$D = 200$[ms]と同様に，図 5.16 に示したように，ナイーブ手法 2 と比べて，DAMCREM とレベルベース動的手法は負荷が高くなった際にもレイテンシを低く保ち，負荷が低くなった際には，レイテンシをより低くすることを達成できた．

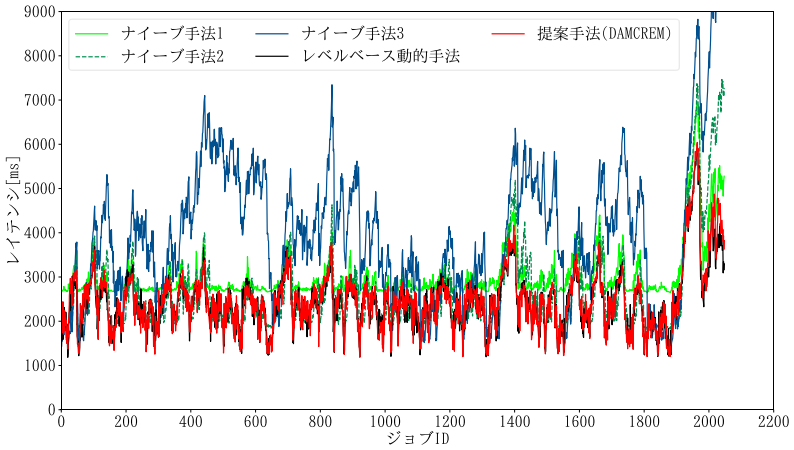

図 5.16　ジョブごとのレイテンシ（ニューラルネットワークにおける推論処理，平均クエリ到着間隔$D = 240$[ms]，11 セット目）

平均クエリ到着間隔$D = 280$[ms]

いずれの手法もデッドラインを守ることができた．DAMCREM とレベルベース動的手法は，ナイーブ手法 3 と比べて中央値が大きい．DAMCREM とレベルベース動的手法では，レイテンシの最小値が小さいため，ナイーブ手法 3 よりも多くのスレッドを割り当てたことにより，実行待ちマクロタスクに割り当てるスレッド数が不足し，少ないスレッドで実行するといったことが生じたと考えられる．ナイーブ手法 4 のレイテンシの標準偏差が大きい理由は，クエリ到着頻度とナイーブ手法 4 のスループットとの差が小さいためである．

平均クエリ到着間隔$D = 320$[ms]

いずれの手法もデッドラインを守ることができた．ナイーブ手法 3 が最もレイテンシの平均値と中央値が小さい．一方，ナイーブ手法 4 は，ナイーブ手法 3 よりもレイテンシの最小値が小さく，DAMCREM とレベルベース動的手法と差が3[ms]以下である．したがって，DAMCREM とレベルベース動的手法は，1 つのマクロタスクに多数のスレッドを割り当ててしまい，使用可能なスレッド数が減少し，少ないスレッドを割り当てたマクロタスクが存在していると考えられる．つまり，割り当てるスレッド数が安定していない．しかし，DAMCREM とレベルベース動的手法は，レイテンシの最大値がナイーブ手法 1〜4 よりも低く保つことができた．

平均クエリ到着間隔$D = 360$[ms]

いずれの手法もデッドラインを守ることができた．ナイーブ手法 3 が最もレイテンシの平均値が小さいのに対し，レイテンシの中央値は，ナイーブ手法 4 が最も小さい．これは，クエリの到着間隔が短くなり，負荷が高くなった時に，ナイーブ手法 4 のレイテンシがナイーブ手法 3 よりも大きく増加したためである．DAMCREM とレベルベース動的手法は平均クエリ到着間隔$D = 320$[ms]と比べて，標準偏差が小さくなっていない．これは，平均クエリ到着間隔$D = 320$[ms]と同様に，使用スレッド数が安定していないことが原因として考えられる．

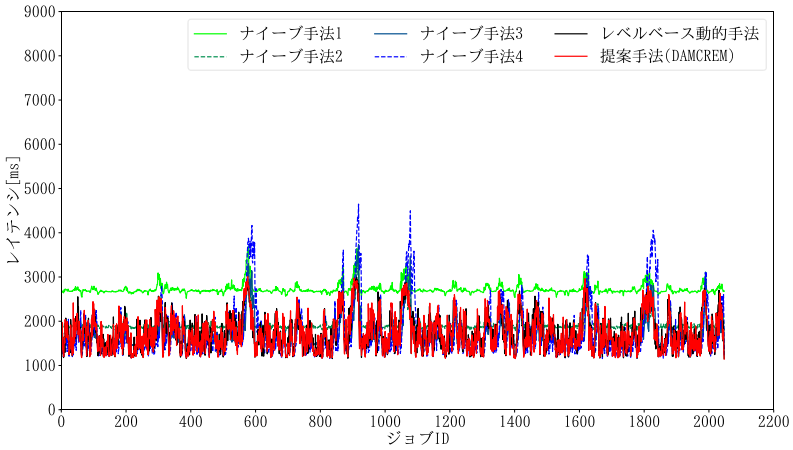

図 5.17　ジョブごとのレイテンシ（ニューラルネットワークにおける推論処理，平均クエリ到着間隔$D = 360$[ms]，11 セット目）

5.5.3　実験結果のまとめ

平均クエリ到着間隔間でのレイテンシの平均値の比較を図 5.18 と表 5.19 に示す．ニューラルネットワークにおける推論処理を対象アプリケーションとした実験では，平均クエリ到着間隔$D = 200$[ms]のように負荷が高い場合は，DAMCREM とレベルベース動的手法は，ナイーブ手法と比べてレイテンシを低くすることができた．一方，平均クエリ到着間隔が280[ms]や320[ms]のように長くなると，レイテンシの平均値がナイーブ手法 3 より 0.5〜4.5%長くなった．多項式近似アプリケーションと同様に，実行待ちマクロタスクのために残しておくスレッド数の見積もりの精度が不十分であり，使用スレッド数が安定していないことが理由として考えられる．

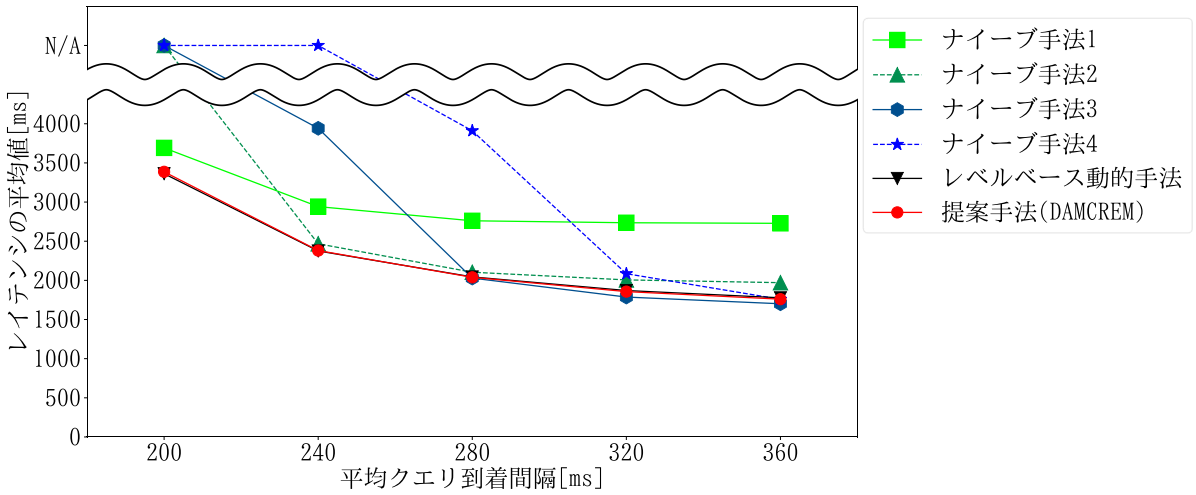

図 5.18　平均クエリ到着間隔とレイテンシの平均値の関係
（ニューラルネットワークにおける推論処理）

※「N/A」は，ジョブのレイテンシがデッドラインである22,758[ms]を超えたことを示す

表 5.19　平均クエリ到着間隔とレイテンシの平均値の関係
（ニューラルネットワークにおける推論処理）

手法	平均クエリ到着間隔D[ms]に対するレイテンシの平均値				
	$D = 200$	$D = 240$	$D = 280$	$D = 320$	$D = 360$
ナイーブ手法 1	3,692	2,942	2,761	2,736	2,728
ナイーブ手法 2	—	2,467	2,104	2,006	1,970
ナイーブ手法 3	—	3,942	2,029	1,787	1,701
ナイーブ手法 4	—	—	3,911	2,085	1,758
レベルベース動的手法	3,362	2,376	2,044	1,870	1,777
DAMCREM	3,386	2,381	2,039	1,856	1,760

※「—」は，ジョブのレイテンシがデッドラインである22,758[ms]を超えたため除外した．

第6章　　考察

本章では，評価実験に対する考察を述べる．

6.1 DAMCREM とナイーブ手法との比較

DAMCREM では，クエリ到着間隔が長くなっても，ジョブのレイテンシの標準偏差がナイーブ手法よりも大きい．もし，スレッド数が安定しているのであれば，ナイーブ手法のように，特定のレイテンシ付近のジョブ数が多くなる．したがって，マクロタスクごとに割り当てるスレッド数が安定していないことが挙げられる．具体的には，あるジョブのマクロタスクには多くのスレッドを割り当てたことにより，低レイテンシを達成できた一方で，別のジョブのマクロタスクには少ないスレッドしか割り当てなかったために，レイテンシを十分短くできなかったということである．原因の一つとして，実行待ちマクロタスクに残しておくべきスレッド数の見積もりが不正確であることが考えられる．

6.2 DAMCREM とレベルベース動的手法の比較

DAMCREM は，レベルベース動的手法と比べて，1 つのマクロタスクに多くのスレッドを割り当てにくい．理由は，レベルベース動的手法は，実行待ちマクロタスクに対して，最大 2 スレッドまでしか残しておかないのに対し，DAMCREM では，レベルベース動的手法よりも多くのスレッドを実行待ちマクロタスクに対して残すためである．したがって，レベルベース動的手法は，提案手法と比べて，実行待ちマクロタスクに残しておくスレッド数が同じか少なくなる．そのため，DAMCREM では，実行待ちマクロタスク数が増加すると，実行待ちマクロタスクのために残しておくスレッド数の見積もりが不適切となっていることが考えられる．具体的には，マクロタスク実行開始時おける，実行待ちマクロタスクリストに対する同期やスレッド生成のオーバヘッドが存在するため，複数のマクロタスクを同時に実行開始できない．したがって，実行待ちマクロタスクのために残しておいたスレッドが，対象のマクロタスクが実行されるまで使用されないという問題である．例えば，評価実験における多項式近似での平均クエリ到着間隔110[ms]のように，常に多数のスレッドを割り当てるのが最もレイテンシを短縮できる場合において，DAMCREM とレベルベース動的手法はレイテンシを十分に短縮できていない．

6.3 アプリケーションの違いに対する考察

多項式近似アプリケーションとニューラルネットワークにおける推論処理アプリケーションの

どちらに対しても，負荷が高い状況では，DAMCREM とレベルベース動的手法は，ナイーブ手法よりもレイテンシを低く保つことができている．そして，負荷が低い状況では，ナイーブ手法 3 やナイーブ手法 4 がレイテンシの平均値や中央値を低く保っている．

負荷が低い条件での DAMCREM とレベルベース動的手法を比較する．多項式近似アプリケーションでは，レイテンシの平均値や中央値はレベルベース動的手法の方が小さい．一方，ニューラルネットワークにおける推論処理アプリケーションでは，レイテンシの平均値や中央値は DAMCREM の方が小さい．これは，DAMCREM の方がレベルベース動的手法よりも多くのスレッドを割り当てにくいため，ニューラルネットワークにおける推論処理でのマクロタスク数の増加に対応できたためと考えられる．

しかし，アプリケーションに対する違いについては，クエリの暗号文のパラメータや多項式近似の次数，ニューラルネットワークの各層のノード数を変更するなど，さらに多くの種類のアプリケーションでの実験が必要である．

第7章　　今後の課題

提案手法 DAMCREM の改良について

6.1 節及び 6.2 節で示した通り，DAMCREM では，実行待ちマクロタスクに残しておくスレッド数をより正確に見積もる必要がある．例えば，マクロタスクの実行開始時における実行待ちマクロタスクリストの同期やスレッド生成によるオーバヘッドを考慮する必要がある．また，DAMCREM では，マクロタスクグラフの構造を使用していないため，実行中マクロタスクのそれぞれの終了時刻や依存関係を用いることが挙げられる．ただし，使用スレッド数の動的選択に必要な計算量が増加するため，いかにオーバヘッドを小さく保つかが解決すべき問題となる．DAMCREM は，単一種類のアプリケーションのみを対象としている．また，クエリである暗号文のレベルや多項式環の次数といったパラメータもジョブ間で同一の値としている．したがって，より汎用性を高めるためには，ジョブごとに異なる暗号文パラメータの暗号文をクエリとする場合や，複数種類のアプリケーションを同時に実行する場合にも対応できるようにする必要がある．ここで，ジョブ間での各マクロタスクに対する優先度設定や使用スレッド数の動的選択に必要な最悪計算量の削減に対して，具体的にどのような手法を用いるかが解決すべき問題となる．例えば，ジョブ間での各マクロタスクに対する優先度を設定する際に，ジョブのデッドラインを用いるといったことが挙げられる．

評価実験について

本評価実験では，DAMCREM に対して，ナイーブ手法とレベルベース動的手法との比較のみを行った．したがって，計算資源を動的にスケジューリングする他の既存手法と比べて，DAMCREM がどの程度優れているかを判断する必要がある．また，負荷が高い状況においては，アプリケーションをマクロタスクに分割せず，常にジョブに 1 つのスレッドのみを割り当てて実行するという手法と比較することで，オーバヘッドによる性能劣化を測定することができる．

時間の都合で，クエリ到着時刻s_iは，各平均クエリ到着間隔に対して 1 通りのみしか試していない．したがって，各平均クエリ到着間隔に対して，より多くのクエリ到着時刻s_iを生成し，実験することで，より誤差の小さい評価を行う必要がある．

評価実験において，平均クエリ到着間隔は試行ごとに定数とした．したがって，平均クエリ到着間隔をジョブ ID もしくは時刻に応じて変化させることで，頻繁に負荷が変化する状況における DAMCREM の性能を評価し，さらなる改善を実施する必要がある．

評価実験において，多項式近似とニューラルネットワークにおける推論の 2 種類のアプリケーションを用いた．しかし，6.3 節に示した通り，アプリケーション間における違いよって生じる，手法間の違い詳しく調べるためには，より多くの種類のアプリケーションを対象に実験を行う必要がある．そして，DAMCREM が適しているもしくは適していないアプリケーションが存在するかを確かめ，適していないアプリケーションが存在すれば，DAMCREM がレイテンシを低く保つことができるように DAMCREM を改良することが課題である．

準同型演算ライブラリについて

提案手法における，使用スレッド数選択処理の最悪計算量は，使用スレッド数の候補数に依存する. 今回は SEAL ライブラリを用いたため, SEAL ライブラリの実装による影響を受ける. SEAL ライブラリは，元々準同型演算を構成する処理を並列化していない．したがって，並列化に適した実装を行うことで，並列化の効果が高いスレッド数のみを厳選して候補とすることで，DAMCREM の性能向上を期待できる.

第8章　　おわりに

本論文では，完全準同型暗号を用いたクライアント—サーバアプリケーションにおける，ジョブの平均レイテンシ短縮を目的に，マクロタスクごとに割り当てるスレッド数を動的に選択する手法 DAMCREM を提案した．評価実験では，負荷が高い状況において，ナイーブ手法よりもレイテンシを低く保つことができた．一方，負荷が低い状況では，常に多数のスレッドを割り当てるナイーブ手法よりもレイテンシが高かった．今後の課題として，使用スレッド数動的選択におけるオーバヘッドを低く保ちつつ，実行待ちマクロタスクのために残しておくスレッド数の見積もりに対する精度を向上させることや，既存のタスクスケジューリング手法と比較を行い，DAMCREM を改良することが挙げられる．

謝辞

本研究及び論文執筆を行なうにあたり，多くのご指導をしてくださった山名早人教授に深く感謝いたします．また，ご指摘やご助言をくださった山名研究室のメンバーや，ニュージャージ工科大学の Andrew Sohn 准教授や Mehtab Sidhu 氏，Yongjian Wang 氏に深く感謝いたします．

本研究は，JST CREST JPMJCR1503 の支援を受けたものである．

参考文献

[1] 大場みち子, “Webサービス:Webサービスの本格的な活用と普及に向けて, ”情報処理, vol.46, no.6，pp.697–700，2005.

[2] I. Anagnostopoulos, V. Tsoutsouras, A. Bartzas and D. Soudris, “Distributed run-time resource management for malleable applications on many-core platforms,” Proc. DAC 2013, no.168, pp1–6, 2013.

[3] V. Tsoutsouras, S. Xydis and D. Soudris, “Application-arrival rate aware distributed run-time resource management for many-core computing platforms,” IEEE Trans. Multi-Scale Comput. Syst. vol.4, no.3, pp.285–298, 2018.

[4] J. Ren, X. Su, G. Xie, C. Yu, G. Tan and G. Wu, “Workload-Aware Harmonic Partitioned Scheduling of Periodic Real-Time Tasks with Constrained Deadlines,” Proc. DAC 2019, no.167, pp.1–6, 2019.

[5] W. Dai and B. Sunar, “cuHE: A homomorphic encryption accelerator library,” Int. Conf. BalkanCryptSec 2015, LNCS, vol.9540, pp.169–186, 2016.

[6] A.A. Badawi, B. Veeravalli, C.F. Mun and K.M.M. Aung, “High-Performance FV Somewhat Homomorphic Encryption on GPUs: An Implementation using CUDA,” IACR Trans. Cryptogr. Hardw. Embed. Syst., vol.2018, issue 2, pp.70–95, 2018.

[7] A. Al Badawi, J. Chao, J. Lin, C. F. Mun, J. J. Sim, B. H. M. Tan, X. Nan, K. M. M. Aung, and V. R. Chandrasekhar, “The AlexNet Moment for Homomorphic Encryption: HCNN, the First Homomorphic CNN on Encrypted Data with GPUs,” Cryptology ePrint Archive: Report 2018/1056, 2018.

[8] M. S. Riazi, K. Laine, B. Pelton, and W. Dai, “HEAX: High-Performance Architecture for Computation on Homomorphically Encrypted Data in the Cloud,” Cryptology ePrint Archive: Report 2019/1066, 2019.

[9] 佐藤宏樹，馬屋原昂，石巻優，山名早人，“完全準同型暗号による秘密計算回路のループ最適化と 最近傍法への適用, ” 第 9 回データ工学と情報マネジメントに関するフォーラム（DEIM2017），no.H6-3，2017.

[10] F. Boemer, Y. Lao, R. Cammarota and C. Wierzynski, “nGraph-HE: A Graph Compiler for Deep Learning on Homomorphically Encrypted Data,” Cryptology ePrint Archive: Report 2019/350, 2018.

[11] F. Boemer, A. Costache, R. Cammarota and C. Wierzynski, "nGraph-HE2: A High-Throughput Framework for Neural Network Inference on Encrypted Data," Cryptology ePrint Archive: Report 2019/947, 2019.

[12] E. Crockett, C. Peikert, and C. Sharp, “Alchemy: A language and compiler for homomorphic encryption made easy,” Proc. CCS 2018, pp.1020–1037, 2018.

[13] S. Carpov, P. Dubrulle, and R. Sirdey, “Armadillo: A compilation chain for privacy preserving applications,” Proc. SCC 2015, pp.13–19, 2015.

[14] V. Lyubashevsky, C. Peikert, and O. Regev, “On Ideal Lattices and Learning with Errors over Rings,” J. ACM, vol.60, issue 6, No.43, 2013.

[15] J.H. Cheon, A. Kim, M. Kim and Y. Song, “Homomorphic Encryption for Arithmetic of Approximate Numbers,” Proc. ASIACRYPT 2017, LNCS, vol.10624, pp.409–437, 2017.

[16] J.H. Cheon, K. Han, A. Kim, M. Kim and Y. Song, Bootstrapping for approximate homomorphic encryption, Proc. EUROCRYPT 2018, LCNS, vol.10820, pp.360–384, 2018.

[17] J.H. Cheon, K. Han, A. Kim, M. Kim and Y. Song, “A Full RNS Variant of Approximate Homomorphic Encryption,” Proc. SAC 2018, LNCS, vol.11349, pp.347–368, 2019.

[18] C. Gentry, S. Halevi, and N. P. Smart, “Homomorphic evaluation of the AES circuit”, CRYPTO 2012, LNCS, vol.7417, pp.850–867, 2012.

[19] “Microsoft SEAL (release 3.3),” https://github.com/Microsoft/SEAL, 2019.

[20] T. Lepoint and P. Paillier, “On the Minimal Number of Bootstrappings in Homomorphic Circuits,” Proc. FC 2013, LNCS, vol.7862, pp.189–200, 2013.

[21] 佐藤宏樹，石巻優，山名早人，“完全準同型暗号における bootstrap problem 及び relinearize problem の厳密解法の高速化, ”第 11 回データ工学と情報マネジメントに関するフォーラム（DEIM2019），no.I5-4，2019.

[22] 鈴木拓也，石巻優，山名早人，“メニーコア CPU 環境における準同型暗号演算高速化を目的とするタスクスケジューリング手法の検討,”研究報告ハイパフォーマンスコンピューティング（HPC），vol.2019-HPC-1，no.28，2019.

[23] A.K. Singh, P. Dziurzanski and L.S. Indrusiak, “Market-inspired dynamic resource allocation in many-core high performance computing systems,” Int. Conf. on High Performance Computing and Simulation, HPCS 2015, pp.413–420, 2015.
[24] 小幡元樹，白子準，神長浩気，石坂一久，笠原博徳，“マルチグレイン並列処理のための階層的並列性制御手法，”情報処理学会論文誌，vol.44，no.4，pp.1‐12，2003.
[25] M. Lowinski, D. Ziegenbein, and S. Glesner, “Splitting tasks for migrating real-Time automotive applications to multi-core ECUs,” Proc. SIES 2016, pp.1–8, 2016.
[26] S. Senobary and M. Naghibzadeh, “First-Fit Semi-partitioned Scheduling Based on Rate Monotonic Algorithm,” Proc. ICCD 2014, AISC, vol.309, pp.173–181, 2015.

研究業績

主著

（1） 鈴木拓也，佐藤宏樹，山名早人，“Knights Landing 上での準同型暗号データ高速処理手法の検討,”第 11 回データ工学と情報マネジメントに関するフォーラム （DEIM 2019），2019.

（2） 鈴木拓也，石巻優，山名早人，“メニーコア CPU 環境における準同型暗号演算高速化を目的とするタスクスケジューリング手法の検討,”研究報告ハイパフォーマンスコンピューティング（HPC），vol.2019-HPC-1，no. 28，2019.

共著

（1） Yusheng Jiang, Tamotsu Noguchi, Nobuyuki Kanno, Yoshiko Yasumura, Takuya Suzuki, Yu Ishimaki and Hayato Yamana, “A Privacy-Preserving Query System using Fully Homomorphic Encryption with Real-World Implementation for Medicine-Side Effect Search,” Int. Conf. iiWAS 2019, 2019.

（2） Rumi Shakya, Yoshiko Yasumura, Suzuki Takuya, Yu Ishimaki and Hayato Yamana, “Outsourced Private Set Union on Multi-Attribute Datasets for Search Protocol using Fully Homomorphic Encryption,” Int. Conf. iiWAS 2019, 2019.